SF 1008-RA (FRONT 9-2003)
Supersedes (8-2003) issue

SAND 2003-3609

REVIEW & APPROVAL FORM

Originating organization: Please complete Sections 1 - 6. Print or type all information. See attached instruction sheets for additional information.

This form is used to review and approve information releases before they are released outside of Sandia.

Public releases must go through the Formal R&A Process, in which case this form must be completed through Section 10.

For releases going through the Organizational R&A Process, organizational management is encouraged to complete this form through Section 6 and to file it for future reference.

This form can also be used (through Section 6 & Section 8) to document approvals when an information release changes distribution limitations (e.g., when Internal Distribution Only becomes Unlimited Release).

For information on which R&A process to use, or additional R&A information:

- See the "Review and Approval for Communication" webpage: http://www-im.sandia.gov/recordsmgmt/revapprov/revapprov.htm or
- Contact Linda Cusimano (NM), (505) 844-4980 ; Kelly McClelland (CA), (925) 294-2311

SECTION 1. Protecting Sandia and Partnership Interests.

Is this release the result of ☐ a CRADA? ☐ Work for Others? ☐ Other partnership, MOU agreement, funding source, or understanding OF ANY KIND? ☐ Information controlled by other agencies

☒ If NO: Go to Section 2.

☐ If YES: Agreement Number is

Has your partner, non-SNL information owner, or funding agency given approval for this release? ☐ Yes ☐ No

SECTION 2. Document Title and Author Information:

Full title of document **The Sandia Petaflops Planner**

Author or contact (Sandian) **Erik P. DeBenedictis** (Full First Name / Full Middle Name / Full Last Name) Signature

Phone No. **505 284 4017** E-Mail Address **epdeben@sandia.gov** Org. No. **9223** Mail Stop No. **1110**

Project number **7101** Task number **13.04** (Identifies funding source – will not be charged)

☐ Contract Author to Sandia. (Contractor's name and contract no.)

SECTION 3. Document Format and Release Event Information. Indicate the planned format(s) of the information release, as well as information about the release event.

Document Format(s):
☒ SAND Report
☐ Abstract
☐ Conference Paper
☐ Journal Article
☐ Sandia Open Network (External)
☒ Computer Software
☐ Publication (all other types of publications including reports, vugraphs, posters, exhibits, displays, videos, brochures, internal memoranda, newsletters, factsheets)

Release Event: Indicate the name of the conference, meeting, or publication, the sponsoring organization, and the place and date of event. If this release is an electronic posting, provide the current viewing address and intended posting location.

Name of Conference / Journal / Book:

Place of Event: Date: **thru**

Internet Address of Electronic Posting:

SECTION 4. Classification and Sensitivity of Information. Contact Classification Dept. 4225 (8511 - CA) for questions.

Indicate classification level and category of information release or whether information release is unclassified:

Classification of: Document Title U Document Abstract U The Document U

☐ Classified - Limited Release. Indicate additional access restrictions:
☐ NWD Sigma ☐ CNWDI ☐ NOFORN ☐ Program Designated Special Handling (Distribution Limitation) ☐ Other

☐ Unclassified - Limited Release. Indicate all Unclassified Controlled Information (UCI) categories access restrictions:
☐ Export Controlled Information (ECI) ITAR/EAR/
☐ Non-Sandia Proprietary Information
☐ Official Use Only (OUO) Exemption No.
☐ Patent Caution
☐ Protected CRADA Information (Release date)
☐ Program Designated Special Handling (Distribution Limitation)
☐ Unclassified Controlled Nuclear Information (UCNI)
☐ Other (specify)

☒ Unclassified - Unlimited Release. Information is unclassified with no access restrictions, i.e., distribution may be made worldwide.

☐ **DUSA Exemption** - This information is released under DUSA Exemption (Mark appropriate sensitivity above. **Section 8 review not required.**)

Derivative Classifier (DC) who is knowledgeable of information sensitivity or DUSA Delegate:

Paul Yarrington (Name) Signature 9223 (Org.) 9/25/03 (Date)

(Continued on page 2)

SF 1008-RA (9-2003)
Supersedes (8-2003) issue

SECTION 5. Disclosure of Technical Advance

A Technical Advance is an original achievement or non-obvious progress in a scientific or engineering sense, including the creation of software. It may be protected by patent or copyright. The Originators of a Technical Advance may be inventors or authors.

Does the subject of this Information Release represent a Technical Advance as defined above?
☐ Yes ☒ No If No, go to Section 6.

If Yes, has a Disclosure of Technical Advance (TA), Form SF 1155-TD, been filed with the Sandia Patent and Licensing Center?
☐ Yes SD No. _____ ☐ No If No, please follow up with a TA form obtainable from:

SECTION 6. Line/Program Signatures and Approvals. Print or type all author information; obtain appropriate signatures from next-level manager. Where concurrence is obtained in case of multiple authors, approval need only go through the principal author's line organization.

Authors' Names (Print or type) (Full First, Middle, & Last Name)	Org. No./ Mail Stop	Phone No.	Next Level Manager's Signature	Date
Erik P. DeBenedictis (Full First Name Full Middle Name Full Last Name)	0223/1110	505 284 4017	Neil Pundit	9/19/03
(Full First Name Full Middle Name Full Last Name)				
(Full First Name Full Middle Name Full Last Name)				
(Full First Name Full Middle Name Full Last Name)				
(Full First Name Full Middle Name Full Last Name)				

Program Manager's Name and Signature _____

THE FOLLOWING SECTIONS ARE USED FOR THE FORMAL REVIEW & APPROVAL PROCESS.

SECTION 7. To be completed by the Patent and Licensing Department (NM: 11500/MS 0161; CA: 11600/MS 9031).

Copyright Interest?	☐ Yes	☒ No	If Yes, copyright may be asserted, subject to DOE approval.
Patent Interest?	☐ Yes	☒ No	If Yes, TA form has been or should be submitted.
Patent Caution?	☐ Yes	☐ No	If Yes, TA form has been or should be submitted and dissemination will be limited.

Patent Attorney's/Agent's Signature _____ Date 10-1-03

SECTION 8. To be completed by the Classification and Sensitive Information Department (NM: 4225/MS 0175, CA 8511, MS 9021).

Signature _____ Date 10/2/03

SECTION 9. To be completed by the Promotional Communications Departments (NM: 9612/MS 0612, CA: 8815/MS 9021).

Promotional Communications must be reviewed for adherence to Corporate "Common Look and Feel" guidelines by PR & Communications Center 12600, MS 0619 (NM) or by Communications & Public Affairs Dept. 8528, MS 9131 (CA). In addition, all publications (including SAND Reports and fact sheets) distributed outside of Sandia that use color (ink) printing, as well as all internal products of two or more colors that use color printing, must go through a Section 9 review. NOTE: SAND Reports and internal release products that use color copying do not need a Section 9 review. (Contact Printing & Duplicating Dept. 12630 to get a head start on DOE approval.)

DOE approval received _____ Date Signature _____ Date _____

SECTION 10. To be completed by the Review & Approval Desk (NM: 9612/MS 0612, CA: 8511/MS 9021).

Approved under the following conditions:

1. That the following statement is printed on the document: **Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.**
2. That, if the release has been determined to be sensitive or classified, all the appropriate markings are on the released item;
3. That all edits requested by reviewers are completed before release;
4. That the final version is submitted to the Technical Library.

Signature Dorothy Martin Date OCT 7 2003

3609
SAND2003-~~####~~
Unlimited Release
Printed ~~August~~ October 2003

The Sandia Petaflops Planner

Erik P. DeBenedictis
Scalable Computing Systems
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-1110

Abstract

The Sandia Petaflops Planner is a tool for projecting the design and performance of parallel supercomputers into the future. The mathematical basis of these projections is the International Technology Roadmap for Semiconductors (ITRS, or an detailed version of Moore's Law) and DOE balance factors for supercomputer procurements. The planner is capable of various forms of scenario analysis, cost estimation, and technology analysis. The tool is described along with technology conclusions regarding PFLOPS-level supercomputers in the upcoming decade.

SAND2003-3609
Unlimited Release
Printed October 2003

The Sandia Petaflops Planner

Erik P. DeBenedictis
Scalable Computing Systems
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-1110

Abstract

The Sandia Petaflops Planner is a tool for projecting the design and performance of parallel supercomputers into the future. The mathematical basis of these projections is the International Technology Roadmap for Semiconductors (ITRS, or a detailed version of Moore's Law) and DOE balance factors for supercomputer procurements. The planner is capable of various forms of scenario analysis, cost estimation, and technology analysis. The tool is described along with technology conclusions regarding PFLOPS-level supercomputers in the upcoming decade.

Intentionally Left Blank

Contents

Figures

Tables

Nomenclature

ASCI…… Advanced Simulation and Computing (the “I” is silent)
ASIC……Application Specific Integrated Circuit
DOE….. .....Department of Energy
DRAM…….....Dynamic Random Access Memory
FLOPS……. .....Floating Operations Per Second
GFLOPS……..... 10^9 Floating Operations per Second
IPC IO….. .....Interprocessor Communications Input Output (I. e. MPI)
ITRS…… International Technology Roadmap for Semiconductors
MPI……..Message Passing Interface
MPP……. ..... Massively Parallel Processor
MPU……. .....MicroProcessor Unit
PetaFLOPS……. ..... 10^{15} Floating Operations Per Second
PIM…….....Processor In Memory
SNL…… .....Sandia National Laboratories
TFLOPS…… ..... 10^{12} Floating Operations Per Second

Intentionally Left Blank

Introduction

Over the past 20 years, parallel supercomputers have gone from an academic curiosity to vital tools for science, defense, engineering, and a host of other fields. The Department of Energy's (DOE's) Advanced Simulation and Computing (ASCI) program has driven the development of parallel supercomputers through a staged series of procurements up to the current programmatic limit of 100 TFLOPS. It seems evident that applications will continue to want more power, causing us to wonder if the computer technology is up to the task?

To answer this, we need to find an oracle to foretell the future of supercomputers. We found our oracle in Moore's Law, or more specifically the extensive body of knowledge known as the ITRS roadmap developed by the semiconductor industry projecting future progress in integrated circuits. To apply Moore's Law for integrated circuits to supercomputers, we need an abstracted definition of a supercomputer that can be applied to hypothetical supercomputers in the future. We found this definition in the procurement rules for DOE supercomputers. Over the half-dozen generations of DOE supercomputer procurements, DOE labs have figured out how to specify the relationships between a supercomputer's computing rate, memory size, communications rate, etc. needed for a machine to work well for applications. Adding Moore's Law to DOE's procurement rules yields hundreds of parameters and mathematical relationships – but which can be solved with effort.

We wrote a "Petaflops Planner" program that essentially projects popular supercomputer designs into the future using Moore's Law. We wrote the planner as a tool for multiple purposes: testing hypotheses on computer architecture, optimizing designs, projecting costs, etc.

We then used the planner to see if computer technology was up to the task of PFLOPS-level supercomputers. The body of the paper explains how the answer can be "yes," but only by employing new technology.

Previous Work

Balance factors

Sandia and other DOE laboratories have had criteria for specifying supercomputers in procurements. These criteria have been applied to supercomputers with vastly different performance levels over the last 20 years. As a result, the criteria have become scalable rules that define supercomputers that will perform well within the DOE.

	DATA-INTENSIVE (RED STORM[1])	COMPUTE-INTENSIVE (PURPLE[2])
Memory Bytes/FLOP	1.0 TFLOPS$^{-.25}$	1.0 TFLOPS$^{-.25}$
Memory Bandwidth Bytes/FLOP	4	1
Peak IPC IO Bytes/sec/FLOP	2	.1/12.≈.01
Total IPC IO Bytes/sec/FLOP	2	.1
Network bisection bandwidth Bytes/sec/FLOP	.075	.05

Table 1: Balance Factors

These scalable rules are based on "balance factors"[ref 1, 2]. A DOE laboratory begins by specifying a performance target – originally measured as peak GFLOPS, now peak TFLOPS. The balance factors specify the values of other parameters as a function of the peak FLOPS. DOE procurements tend to use similar balance factors, but with different values representing a different application mix (table 1).

Architectures and Packaging Styles

The supercomputing community has just a few leading architectures or design styles:

- Virtually all supercomputer today are clusters of Symmetric MultiProcessors (SMPs, figure 1), which are composed of separate commodity microprocessor chips, memory chips, and a router implemented as an Application Specific Integrated Circuit (ASIC).

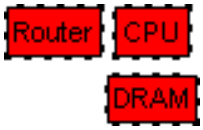

Figure 1: An SMP Node

- nCUBE Corporation pioneered a different implementation style where the CPU and router are both in the same ASIC chip, but with separate memory (figure 2). IBM is using this implementation style for the DOE Blue Light project.

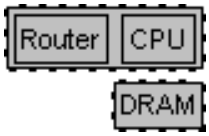

Figure 2: An nCUBE Node

- Processor-In-Memory (PIM) represents a more radical departure. PIMs include processors, memory, and router in a single chip – often hundreds of each (figure 3). PIM systems may consist of homogeneous arrays of PIM chips, but may also include additional conventional memory. There are different views on the best "instruction set" for PIMs, and some designs may represent a different architecture as well as a different packaging style.

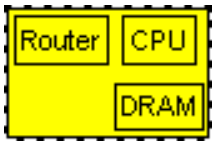

Figure 3: A PIM Node

Semiconductor Roadmap

The ITRS roadmap[ref. 3] is a project by the semiconductor industry to track the progress of semiconductor technology, identifying roadblocks before they become problems. While Moore's Law is a one-sentence statement, the ITRS Roadmap is a time comparable in size to a telephone book (figure 4).

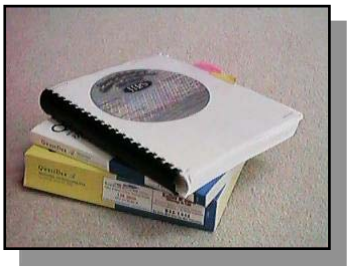

Figure 4: The ITRS compared to the Albuquerque Telephone Book

A sample from the 1999 ITRS roadmap illustrates the process. The Semiconductor Industries Association (SIA) establishes a panel of experts in each of about a dozen aspects of semiconductor technology (process technology, packaging, DRAM design, testing, …). These experts analyze progress in their area, either confirming that Moore's Law can be maintained or identifying obstacles. To fit on the printed page, each table comes in a near term and long term part (figure 5). Each technology item is designated white, yellow,

or red depending on the state of known technology. As illustrated (figure 5), assembly and packaging are areas of concern in 1999.

Table 59b Assembly & Packaging Technology Requirements—Long Term (continued)

YEAR TECHNOLOGY NODE	2008 70 nm	2011 50 nm	2014 35 nm
Performance: On-Chip (MHz) [E]			
Low cost	840	1044	1250
Hand-held	840	1044	1250
Cost-performance	1400	1800	2200
High-performance	2500	3000	3600
Harsh	100	100	100
Memory (D/SRAM)	175/700	200/900	225/1100
Performance: Chip-to-Board for Peripheral Buses (MHz)			
Low cost	125	125	150
Hand-held	125	125	150
Cost-performance [F]	175/700	200/900	225/1100
High-performance [G]	1250	1500	1800
Harsh	100	100	125
Memory (D/SRAM) [F]	175/700	200/900	225/1100

Solutions Exist *Solutions Being Pursued* *No Known Solutions*

Figure 5: Example Chart from ITRS

The Petaflops Planner

Abstract Supercomputer Representation

The planner represents a supercomputer as a collection of SMP nodes. Each node is comprised of chips defined by Silicon area and pin count. The wires connect the pins. The planner then calculates system performance assuming the system is assembled according to on of the architectures described above. The planner assumes each chip, pin, and wire will perform to its maximum capacity. System cost is computed by using ITRS cost figures for ASICs, MPUs, DRAMS, etc.

The illustration of a PIM (figure 6) illustrates the planner's level of abstraction. The rectangle is displayed of a size proportional to the chip area. The yellow area represents RAM on a PIM and blue and red represent processor cores, with the amount of each color proportional to the Silicon area consumed by the component. The planner deals only with area, not specific circuitry.

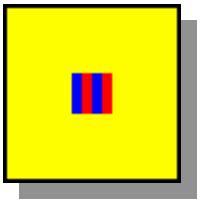

Figure 6: PIM Chip Layout Model

Interconnect

The Petaflops Planner assumes a 3-d mesh topology. The rationale is as follows:

- By choosing a 3-d mesh topology, we trivialize analysis of the mapping between the supercomputer topology and the topology of the machine room. All machine rooms in the universe are 3-d, because the universe is 3-d (spatially). Thus, we can put a 3-d supercomputer into a machine room without having any long wires. The “fat tree” is other popular interconnect topology. Putting a “fat tree” supercomputer into a 3-d machine room involves a spatial mapping and some long wires. Long wires introduce both cost and delay.

- Sandia National Laboratories (SNL) has a history of procuring 3-d mesh supercomputers that remain the “fastest computer on earth” longer than anybody expects. Hence, there is no evidence of a performance penalty by limiting the analysis to 3-d meshes.

Derated Flops

The planner uses a method we call “derated flops” to calculate system performance. The rationale is as follows: Imagine that system performance was based solely on peak FLOPS, and that we would use an architectural optimization method (as we do). With peak FLOPS as the objective function, an optimizer would be expected to find systems where the chips were packed solid with floating point units. These systems would not perform well on real applications because they would lack memory and IPC IO. We found that by “derating” the flop rate so that balance was preserved, we got an objective function suitable for optimization.

We define the “derated flops” for a system as the maximum number of FLOPS that can be claimed while still meeting the balance factors. For example, say the balance factors call for one byte of memory bandwidth per flop. A microprocessor with 10 GFLOPS floating performance and 1 gbyte/second memory bandwidth would therefore be unbalanced. However, we permit the vendor to claim a “derated flops” rate of 1 GFLOP for the microprocessor. One GFLOP is the

highest flop rate that could be claimed without violating the balance factor for memory bandwidth.

Other Factors

We considered two other factors in designing the planner:

Technology derating factor. Observers have long noticed that industry puts the best technology into consumer and commercial products where volumes and profits are highest. Supercomputing tends to get technology a couple years later. To capture this behavior, the planner lets the user specify a “technology derating factor.” Certain ITRS projections get multiplied by this factor to model this delay. The figures that get cut include ASIC density and clock rate, but do not include commodity microprocessor performance and memory density (because supercomputers can include the latest commodity components).

Power. The planner calculates power consumption, but does not use it as a constraint. The calculation applies CMOS power scaling rules to typical values for today’s microprocessors and memories. (It would be desirable for the planner to know maximum power dissipation per chip and then stop allocating components when that power has been reached.)

Architectural Optimization

The planner can optimize designs. The architectural alternatives being considered are parameterized by one or two integers representing the number of CPU cores or memory chips. The planner picks the optimal value(s) for these parameters by iterating over all reasonable values of these integers and noting the value producing the lowest cost per derated flops ratio.

Implementation

We constructed the Petaflops Planner to implement the design process described above. The planner was written in C++ for Windows and Linux and acts as an Internet Web server. That is, the planner starts and listens on a TCP/IP port for Web page accesses from a browser. A user accessing the planner via a browser sees a small Web site with help files, forms for specifying balance factors, and dynamic pages representing candidate supercomputers.

The diagram (figure 7) shows the Web form where the user specifies balance factors.

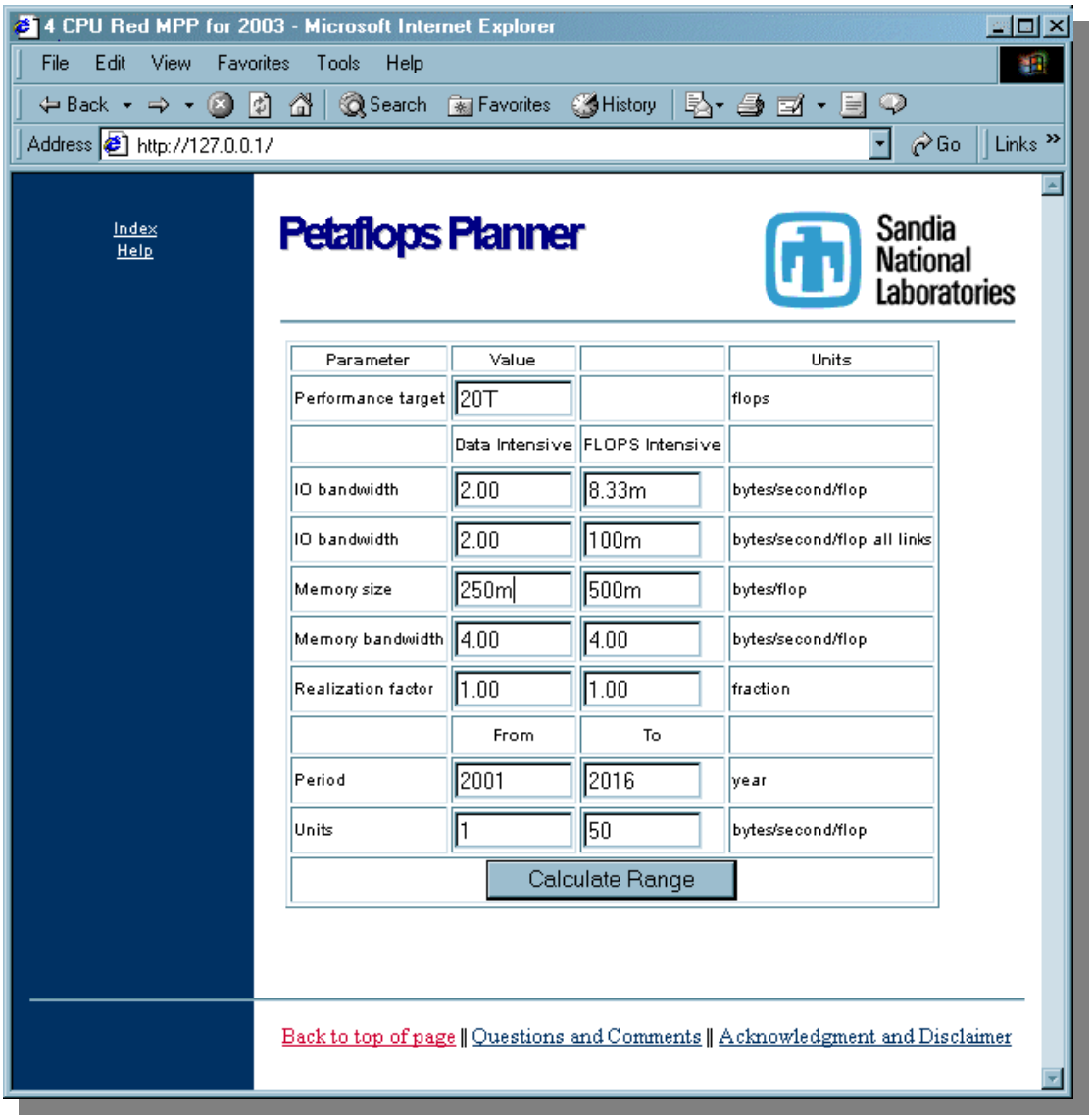

Figure 7: Petaflops Planner Input Form

Upon receiving balance factors, the planner computes several dozen optimal designs and provides a table of hyperlinks for details on each design. The table (not shown) has rows corresponding to year within the range of the ITRS projections and columns corresponding to four candidate architectures and packaging styles. The values in the table cells report the optimal number of CPU cores for the designated architecture in the designated year. Clicking on the hyperlink in the table cell yields a detail page for the design.

The diagram (figure 8) shows a detail page. The planner shows a diagram of a node board, scaling the squares representing chips proportionally to the necessary chip area. The parameters (size, power dissipation, memory capacity)

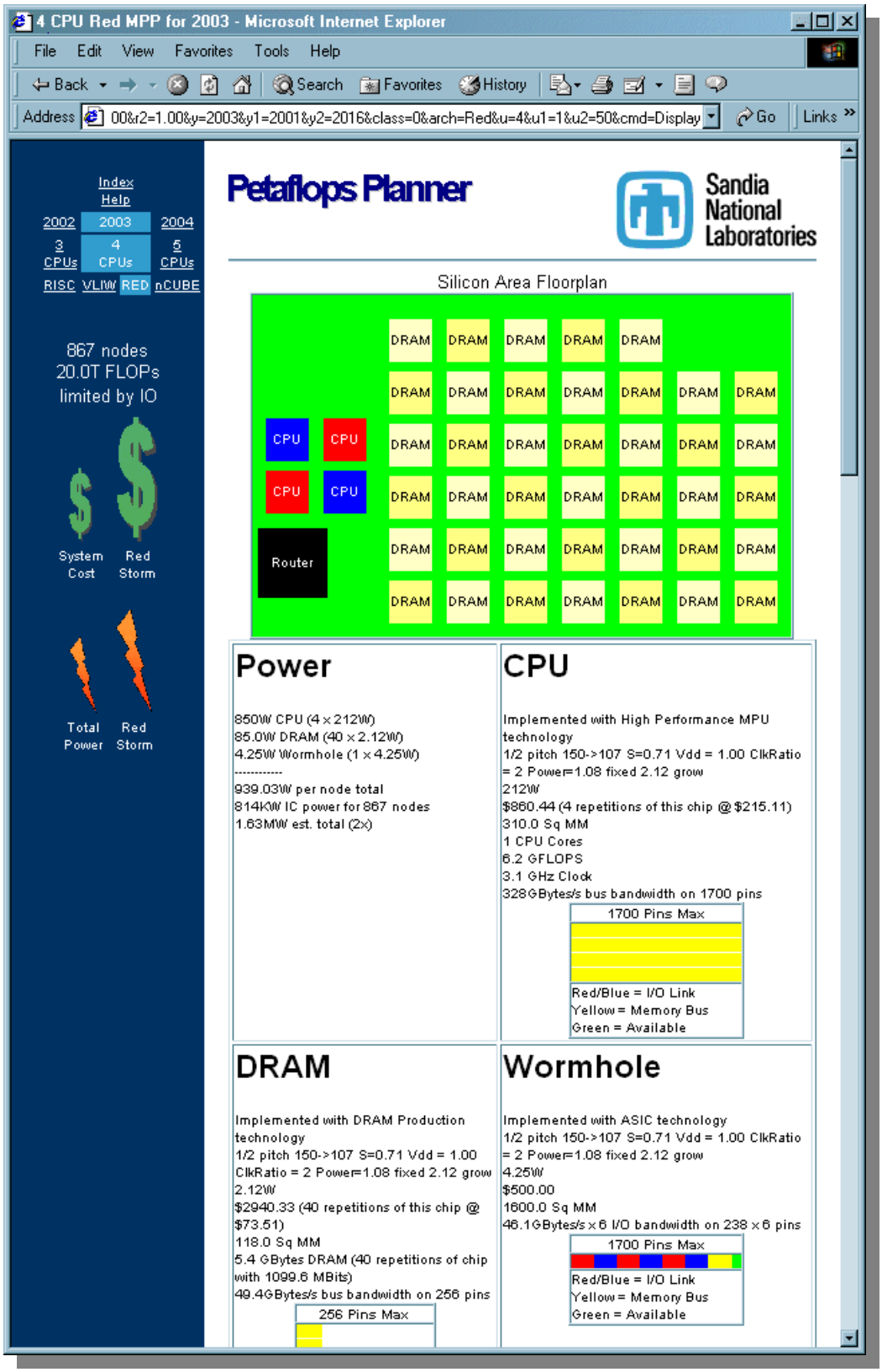

Figure 8: Petaflops Planner Output Page

of each type are reported textually. A bar graph shows the maximum number of pins on a chip and their allocation. The upper-right of the Web page includes a navigation panel that will display detail pages for designs in other years, architectures, and numbers of processors.

Results of Using the Petaflops Planner

The Tool Worked

We have had no problems with stability of the optimization algorithm. Furthermore, we've reviewed the planner's output with various people at Sandia and the results appeared plausible. The tool's speed is acceptable: a 266 MHz PC takes about 2 seconds to perform the ~10,000 design evaluations resulting from submitting balance parameters.

Moore's Law Holds

We wanted to know if supercomputer performance would continue to increase at historical rates to the PFLOPS level. To do this, we calculated and plotted (on a log scale) the estimated cost for an optimal 5 PFLOPS supercomputer for each year into the future and for all the architectures. We expected to see one of two behaviors, both represented in the graph (figure 9):

- Straight lines sloping downward, representing continuation of the exponential progress of Moore's Law.

- Lines that slope downwards for while, followed by a leveling off. This would be the expected result if there were a critical, non-scalable technology component: as other technology components pass it by, the non-scalable component dominates the cost.

As shown in the graph, the planner found both behaviors. The shorter lines (ending in 2014) are based on the 1999 ITRS projections and predict failure of Moore's Law, whereas the longer lines (ending in 2016) are based on 2000 ITRS projections and predict continuation of Moore's Law. This discrepancy is described further below.

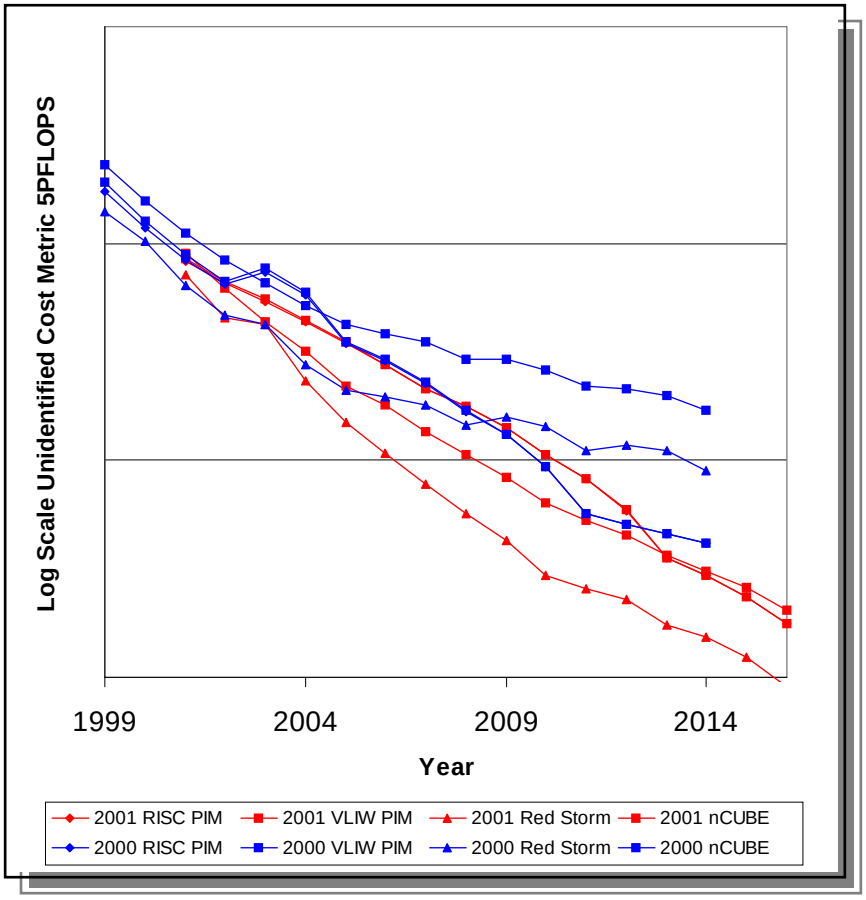

Figure 9: Architecture Costs

Wires and LVDS

The change in conclusion between projections based on 1999 and 2000 ITRS tables illustrates the power and limitations of our approach.

In late 2001 we ran the analysis with the then-current ITRS 1999 tables and were alarmed at the apparent failure of Moore's Law. Examining the optimized designs in the later years revealed enormous chips with thousands of pins with IPC IO busses between chips hundreds of pins wide. Even with such enormous resources applied to chip-to-chip communications, the cores of the chips were "starved" for data. The pertinent ITRS table (figure 5) confirmed the problem: the speed of chip-to-chip communications via pins is limited to several gigabits/second/wire, and the maximum number of pins on a chip is not scaling with Moore's Law. Apparently supercomputers require chip-to-chip bandwidth beyond the scalability of pin-and-wire technology, and as a result Moore's Law fails.

When we ran the planner in early 2002 with the new ITRS 2001 tables, the problem went away. Between 1999 and 2000, a new type of chip-to-chip signaling technology reached the threshold of acceptance and was incorporated into the ITRS 2000 tables. The new technology was Low Voltage Differential Signaling (LVDS, right). Instead of using one wire per bit, LVDS uses two wires driven differentially (+V and -V). The signals are detected by differential amplifiers, where electrical noise cancels. This permits reliable signal recovery at lower power and at higher speed. While LVDS uses twice the pins and wires, the transmission rate becomes scalable with Moore's law. A regular wire is limited to 1 gbit/second whereas a LVDS pair can to 50 gbits/second within the range of projection.

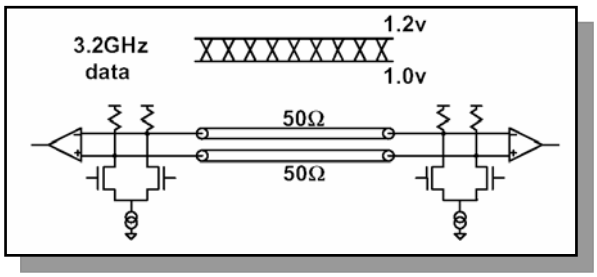

Figure 10: Low Voltage Differential Signaling

No Obvious Winners

No obvious winner emerges from the graph. This is particularly notable because advocates of alternative architectures and packaging styles tout their approaches as being considerably more cost effective than existing systems.

Conclusions

We believe our method has applicability beyond the analysis in this paper. In particular, the ITRS roadmap has adequate predictive power and supercomputer architectures are well enough understood that it is worth the effort to create a roadmap for supercomputer design. Predicting the future is notoriously difficult, as our experience with the change in the ITRS roadmap between 1999 and 2000 illustrated.

We successfully applied optimization techniques to supercomputer architecture. This could be useful in the future.

We conclude that LVDS has a fundamental advantage over pin-and-wire by using a software tool for "scenario analysis" of supercomputer architecture.

There was a widespread belief as recently as 1995, that a petaflop-level computer would be unachievable. This belief is not confirmed.

Future Work

It should be possible to optimize supercomputer for actual applications rather than balance factors. With considerable analysis, one can model the runtime of some “computational kernels” on the abstract hardware like that produced by the planner[ref 4]. The optimizer could then find the architecture that offers the lowest cost per given speed of solution of some application. If this method could be applied to a weighted mix of applications, this would answer the question of “what machine is best for my applications.”

This form of analysis presupposes that computers will be built from the commercial chips covered by the ITRS. It does not account for wafer scale integration or non-CMOS devices. While CMOS technology has a huge maturity advantage over these other technologies, a subsequent analysis should cover them.

References

1. Statement of Work for the Sandia Red Storm Computing Capability Development Program Procurement, RFQ Number 2999, Sandia National Laboratories

2. ASCI Purple RFP, Lawrence Livermore National Laboratory, http://www.llnl.gov/ASCI/purple/

3. 1999 and 2000 International Technology Roadmap for Semiconductors (ITRS), Semiconductor Industries Association (SIA) http://public.itrs.net

4. Predictive Performance and Scalability Modeling of a Large-Scale Application, Darren J. Kerbyson, Henry J. Alme, Adolfy Hoisie, Fabrizio Petrini, Harvey J. Wasserman, Michael L. Gittings, Proceedings of Supercomputing 2001.

Appendix

PIM Design Equations

Optimizer inputs: A node is an SMP with n CPU cores.

The chip area from the ITRS is allocated first to n CPU cores. The remaining chip area is divided by the memory density from the ITRS to get the amount of memory per PIM chip.

The signal pins are divided into 6 bundles, all for IPC IO.

DERATED FLOPS IS MINIMUM OF:	EXPLANATION:
n × peak FLOPS per core	CPU FLOPS cannot exceed raw capability of CPU cores
$\frac{\text{ASIC I/O bandwidth}}{\text{IPC IO balance factor}}$	Derate CPU FLOPS as necessary to balance IPC IO bandwidth
$\frac{\text{memory capacity per core}}{\text{memory capacity balance factor}}$	Derate CPU FLOPS as necessary to balance memory size

Table 2: PIM Design Equations

Power computed by assuming the chip is x% logic and (100-x)% DRAM. The logic and DRAM scale from current levels by CMOS scaling rules.

NCUBE Design Equations

Optimizer inputs: A node is an SMP with n CPU cores in the ASIC and m DRAM chips; the optimizer picks n and m.

The ASIC's signal pins are divided into 7 bundles: 6 of size × for IPC IO and one of size y for the memory bus. y = x × memory bandwidth balance factor / IPC IO balance factor.

Derated Flops is minimum of:	Explanation:
n × peak FLOPS per core	CPU FLOPS cannot exceed raw capability of CPU cores
router chip I/O bandwidth / IPC IO balance factor	Derate CPU FLOPS as necessary to balance IPC IO bandwidth
m × memory capacity per DRAM / memory capacity balance factor	Derate CPU FLOPS as necessary to balance memory size
ASIC bus bandwidth / memory bandwidth balance factor	Derate CPU FLOPS as necessary to balance memory bus bandwidth at CPU
m × memory bus bandwidth / memory bandwidth balance factor	Derate CPU FLOPS as necessary to balance memory bus bandwidth at memories

Table 3: nCUBE Design Equations

ASIC, DRAM and MPU cost from ITRS

Discrete MPP Design Equations

Optimizer inputs: A node is an SMP with n CPU chips and m DRAM chips; the optimizer picks n and m.

Derated Flops is minimum of:	Explanation:
n × peak FLOPS per CPU chip	CPU FLOPS cannot exceed raw capability of CPU cores
router chip I/O bandwidth / IPC IO balance factor	Derate CPU FLOPS as necessary to balance IPC IO bandwidth
m × memory capacity per DRAM / memory capacity balance factor	Derate CPU FLOPS as necessary to balance memory size
CPU bus bandwidth / memory bandwidth balance factor	Derate CPU FLOPS as necessary to balance memory bus bandwidth at CPU
m × memory bus bandwidth / memory bandwidth balance factor	Derate CPU FLOPS as necessary to balance memory bus bandwidth at memories

Table 4: Discrete MPP Design Equations

ASIC $500, DRAM and MPU cost from ITRS

Distribution:

1 MS	9037	J. C. Berry, 8945
1	9019	S. C. Carpenter, 8945
1	9012	J. A. Friesen, 8963
1	9012	S. C. Gray, 8949
1	9011	B. V. Hess, 8941
1	9915	M. L. Koszykowski, 8961
1	9019	B. A. Maxwell, 8945
1	9012	P. E. Nielan, 8964
1	9217	S. W. Thomas, 8962
1	0824	A. C. Ratzel, 9110
1	0847	H. S. Morgan, 9120
1	0824	J. L. Moya, 9130
1	0835	J. M. McGlaun, 9140
1	0833	B. J. Hunter, 9103
1	0834	M. R. Prarie, 9112
1	0555	M. S. Garrett, 9122
1	0821	L. A. Gritzo, 9132
1	0835	E. A. Boucheron, 9141
1	0826	S. N. Kempka, 9113
1	0893	J. Pott, 9123
1	1183	M. W. Pilch, 9133
1	0835	K. F. Alvin, 9142
1	0834	J. E. Johannes, 9114
1	0847	J. M. Redmond, 9124
1	1135	S. R. Heffelfinger, 9134
1	0826	J. D. Zepper, 9143
1	0825	B. Hassan, 9115
1	0557	T. J. Baca, 9125
1	0836	E. S. Hertel, Jr., 9116
1	0847	R. A. May, 9126
1	0836	R. O. Griffith, 9117
1	0847	J. Jung, 9127
1	0321	P. R. Graham, 9208
1	0318	J. E. Nelson, 9209
1	0847	S. A. Mitchell, 9211
1	0310	M. D. Rintoul, 9212
1	1110	D. E. Womble, 9214
1	1111	B. A. Hendrickson, 9215
1	0310	R. W. Leland, 9220
1	1110	N. D. Pundit, 9223
1	1110	D. W. Doerfler, 9224
1	0847	T. D. Blacker, 9226
1	0822	P. Heermann, 9227
1 MS	0818	P. Yarrington, 9230
1	0819	R. M. Summers, 9231
1	0820	P. F. Chavez, 9232
1	0316	S. S. Dosanjh, 9233
1	0316	J. B. Aidun, 9235
1	0813	R. M. Cahoon, 9311
1	0801	F. W. Mason, 9320
1	0806	C. Jones, 9322
1	0822	C. Pavlakos, 9326
1	0807	J. P. Noe, 9328
1	0805	W.D. Swartz, 9329
1	0812	M. R. Sjulin, 9330
1	0813	A. Maese, 9333
1	0812	M. J. Benson, 9334
1	0809	G. E. Connor, 9335
1	0806	L. Stans, 9336
1	1110	R. B. Brightwell, 9224
1	1110	R. E. Riesen, 9223
1	1110	K. D. Underwood, 9223
1	1110	E. P. DeBenedictis, 9223
1	0321	W. Camp, 9200
1	0841	T. Bickel, 9100
1	9003	K. Washington, 8900
1	0801	A. Hale, 9300
1	0139	M. Vahle, 9900
1	9018	Central Technical Files, 8945-1
2	0899	Technical Library, 9616

Software

Note: the C++ software for the Petaflops Planner is included in the document bundle for the Sandia public release only. The actual software would be distributed separately and electronically.

```
// Planner.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"

#include <stdio.h>
#include <math.h>
#include <string.h>

#if __GNUC__ != 0                                                    // Linux
#include <sys/types.h>
#include <sys/stat.h>
#include <malloc.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <errno.h>
#define SOCKET int
#define INVALID_SOCKET -1
#define SOCKET_ERROR -1
int stricmp(const char *a, const char *b) { return strcmp(a, b); }
#define min(a, b) ((a) < (b) ? (a) : (b))
#define max(a, b) ((a) > (b) ? (a) : (b))

#else                                                                // Windows
#include <malloc.h>
#include <winsock.h>
#endif

#define ASSERT(c) if (!(c)) { fprintf(stderr, "assert failed"); fflush(stderr); char *p = NULL; *p++; }

#define BOOL int
#define FALSE 0

#define YEAR 2001

char *Index1 = "<font face=arial size=0>";
char *Index1a = "<font face=arial size=2>";
char *Index2 = "</font>";
char *Out1 = "<font face=arial size=0>";
char *Out1a = "<font face=arial size=2>";
char *Out1x = "";
char *Out1y = " bgcolor=yellow";
char *Out2 = "</font>";
char *Chip1 = "<font size=0 face=arial>";
char *Chip2 = "</font>";
char *DRAMColor1 = "#ffffc0";
char *DRAMColor2 = "#ffff80";
char *PCBColor = "#00ff00";
char *PCBoardGIF = "green.gif";
char *CPUColorA = "blue";
char *CPUColorB = "red";
            char *navbkg = "#003366";
            char *NOut1 = "<font face=arial size=0 color=white>";
            char *NOut1a = "<font face=arial size=2 color=white>";
            char *NOut1x = "";
            char *NOut1y = " bgcolor=#319ace";
            char *NOut2 = "</font>";
            char *navlnk = "white";
            char *navalnk = "blue";
            char *navvlnk = "pink";
int CS = 3;
int CP = 0;
#define SCALE 3.0

// speed of external wire compared to local clock
// .5 corresponds to dual-rail signalling at the local clock rate (full speed, but uses 2 pins per bit)
#define BUSFACTOR .5

char *GreekSuffix(double x) {
            static char dope[20][1000];
            static int next = 0;
            static struct g {
                        double limit;
                        char *format;
                        double divide;
            } table[] = {
                        { 1e-11, "%.2lfp", 1e-12, },
                        { 1e-10, "%.1lfp", 1e-12, },
                        { 1e-9, "%.0lfp", 1e-12, },
                        { 1e-8, "%.2lfn", 1e-9, },
                        { 1e-7, "%.1lfn", 1e-9, },
                        { 1e-6, "%.0lfn", 1e-9, },
                        { 1e-5, "%.2lfu", 1e-6, },
                        { 1e-4, "%.1lfu", 1e-6, },
                        { 1e-3, "%.0lfu", 1e-6, },
                        { 1e-2, "%.2lfm", 1e-3, },
                        { 1e-1, "%.1lfm", 1e-3, },
                        { 1e0, "%.0lfm", 1e-3, },
                        { 1e1, "%.2lf", 1, },
                        { 1e2, "%.1lf", 1, },
                        { 1e3, "%.0lf", 1, },
                        { 1e4, "%.2lfK", 1e3, },
                        { 1e5, "%.1lfK", 1e3, },
                        { 1e6, "%.0lfK", 1e3, },
                        { 1e7, "%.2lfM", 1e6, },
                        { 1e8, "%.1lfM", 1e6, },
                        { 1e9, "%.0lfM", 1e6, },
                        { 1e10, "%.2lfG", 1e9, },
                        { 1e11, "%.1lfG", 1e9, },
                        { 1e12, "%.0lfG", 1e9, },
                        { 1e13, "%.2lfT", 1e12, },
                        { 1e14, "%.1lfT", 1e12, },
```

```
                { 1e15, "%.01fT", 1e12, },
                { 1e16, "%.21fP", 1e15, },
                { 1e17, "%.11fP", 1e15, },
                { 1e18, "%.01fP", 1e15, },
        };
        if (x == 0) {
                sprintf(dope[next%20], "0");
                return dope[next++%20];
        }
        for (unsigned int i = 0; i < sizeof(table)/sizeof(g); i++)
                if (x < table[i].limit) {
                        sprintf(dope[next%20], table[i].format, x/table[i].divide);
                        return dope[next++%20];
                }
        sprintf(dope[next%20], "%.01e", x);
        return dope[next++%20];
}

enum CoreType {
        RISC,
        VLIW,
        RED,
        NCUBE,
};

class Query {
public:
        double PerformanceTarget;
        CoreType Type;
        double IO_BytesPerSecondPerFlop1;   // per link
        double IO_BytesPerSecondPerFlop2;
        double IO_BytesPerSecondPerFlopAll1;// all links together
        double IO_BytesPerSecondPerFlopAll2;
        double RAM_BytesPerFlop1;                                       // memory size
        double RAM_BytesPerFlop2;
        double RAM_BytesPerSecondPerFlop1;  // memory bandwidth
        double RAM_BytesPerSecondPerFlop2;
        double Realization1;                                            // technology realization factor
        double Realization2;
        double Year;
        double Year1;
        double Year2;
        int Class;                                                                      // 0/1 -> 1 vs. 2 for balance factors
        int Units;                                                                      // number of CPU cores or chips
        int Units1;
        int Units2;
        int Command;

        // convert a double, possibly with a GreekSuffix suffix
        void SD(double &d, char *v) {
                sscanf(v, "%lf", &d);
                for (int i = strlen(v); --i >= 0; ) {
                        if (v[i] == ' ') continue;
                        else if (v[i] == 'f') d *= 1e-15;
                        else if (v[i] == 'p') d *= 1e-12;
                        else if (v[i] == 'n') d *= 1e-9;
                        else if (v[i] == 'u') d *= 1e-6;
                        else if (v[i] == 'm') d *= 1e-3;
                        else if (v[i] == 'k' || v[i] == 'K') d *= 1e3;
                        else if (v[i] == 'M') d *= 1e6;
                        else if (v[i] == 'g' || v[i] == 'G') d *= 1e9;
                        else if (v[i] == 't' || v[i] == 'T') d *= 1e12;
                        else if (v[i] == 'P') d *= 1e15;
                        else continue;
                        break;
                }
        }

        // convert an integer, possibly with a GreekSuffix suffix
        void SI(int &n, char *v) {
                double d;
                sscanf(v, "%lf", &d);
                for (int i = strlen(v); --i >= 0; ) {
                        if (v[i] == ' ') continue;
                        else if (v[i] == 'f') d *= 1e-15;
                        else if (v[i] == 'p') d *= 1e-12;
                        else if (v[i] == 'n') d *= 1e-9;
                        else if (v[i] == 'u') d *= 1e-6;
                        else if (v[i] == 'm') d *= 1e-3;
                        else if (v[i] == 'k' || v[i] == 'K') d *= 1e3;
                        else if (v[i] == 'M') d *= 1e6;
                        else if (v[i] == 'g' || v[i] == 'G') d *= 1e9;
                        else if (v[i] == 't' || v[i] == 'T') d *= 1e12;
                        else if (v[i] == 'P') d *= 1e15;
                        else continue;
                        break;
                }
                n = (int)d;
        }

        // construct a query, basing values on a previous submission
        Query(int argc, char **name, char **value) {
                PerformanceTarget = 1e15;
                Type = RISC;
                IO_BytesPerSecondPerFlop1 = 2;      // per link
                IO_BytesPerSecondPerFlop2 = .1/12;
                IO_BytesPerSecondPerFlopAll1 = 2;   // all links together
                IO_BytesPerSecondPerFlopAll2 = .1;
                RAM_BytesPerFlop1 = .5;                         // memory size
                RAM_BytesPerFlop2 = .5;
                RAM_BytesPerSecondPerFlop1 = 4;     // memory bandwidth
                RAM_BytesPerSecondPerFlop2 = 4;
                Realization1 = 1;                                       // technology realization factor
                Realization2 = 1;
                Year = 2010;
#if YEAR == 2000
                Year1 = 1999;
                Year2 = 2014;
#else
                Year1 = 2001;
                Year2 = 2016;
#endif
                Class = 0;
                Units = 6;
```

```
                        Units1 = 1;
                        Units2 = 50;
                        Command = 0;

                        for (int i = 0; i < argc; i++) {
                                char *n = name[i];
                                char *v = value[i];
                                if (strcmp(n, "pt") == 0) SD(PerformanceTarget, v);

                                else if (strcmp(n, "a1") == 0) SD(IO_BytesPerSecondPerFlop1, v);
                                else if (strcmp(n, "a2") == 0) SD(IO_BytesPerSecondPerFlop2, v);

                                else if (strcmp(n, "b1") == 0) SD(IO_BytesPerSecondPerFlopAll1, v);
                                else if (strcmp(n, "b2") == 0) SD(IO_BytesPerSecondPerFlopAll2, v);

                                else if (strcmp(n, "c1") == 0) SD(RAM_BytesPerFlop1, v);
                                else if (strcmp(n, "c2") == 0) SD(RAM_BytesPerFlop2, v);

                                else if (strcmp(n, "d1") == 0) SD(RAM_BytesPerSecondPerFlop1, v);
                                else if (strcmp(n, "d2") == 0) SD(RAM_BytesPerSecondPerFlop2, v);

                                else if (strcmp(n, "r1") == 0) SD(Realization1, v);
                                else if (strcmp(n, "r2") == 0) SD(Realization2, v);

                                else if (strcmp(n, "y") == 0) SD(Year, v);
                                else if (strcmp(n, "y1") == 0) SD(Year1, v);
                                else if (strcmp(n, "y2") == 0) SD(Year2, v);

                                else if (strcmp(n, "class") == 0) SI(Class, v);

                                else if (strcmp(n, "arch") == 0) {
                                        if (strcmp(v, "RISC") == 0) Type = RISC;
                                        else if (strcmp(v, "VLIW") == 0) Type = VLIW;
                                        else if (strcmp(v, "Red") == 0) Type = RED;
                                        else if (strcmp(v, "nCUBE") == 0) Type = NCUBE;
                                }

                                else if (strcmp(n, "u") == 0) SI(Units, v);
                                else if (strcmp(n, "u1") == 0) SI(Units1, v);
                                else if (strcmp(n, "u2") == 0) SI(Units2, v);

                                else if (strcmp(n, "cmd") == 0) {
                                        if (strcmp(v, "Display") == 0) Command = 1;
                                        else if (strcmp(v, "Calculate+Range") == 0) Command = 2;
                                }
                        }
                }

                // output a URL to the connected socket representing this query
                void OutURL(SOCKET s) {
                        char buf[1000];

                        sprintf(buf,
"/planner?pt=%s&a1=%s&a2=%s&b1=%s&b2=%s&c1=%s&c2=%s&d1=%s&d2=%s&r1=%s&r2=%s&y=%.0lf&y1=%.0lf&y2=%.0lf&class=%d&arch=%s&u=%d&u1=%d&u2=%d&cmd=%s"
,
                                GreekSuffix(PerformanceTarget),
                                GreekSuffix(IO_BytesPerSecondPerFlop1), GreekSuffix(IO_BytesPerSecondPerFlop2),
                                GreekSuffix(IO_BytesPerSecondPerFlopAll1), GreekSuffix(IO_BytesPerSecondPerFlopAll2),
                                GreekSuffix(RAM_BytesPerFlop1), GreekSuffix(RAM_BytesPerFlop2),
                                GreekSuffix(RAM_BytesPerSecondPerFlop1), GreekSuffix(RAM_BytesPerSecondPerFlop2),
                                GreekSuffix(Realization1), GreekSuffix(Realization2),
                                Year, Year1, Year2,
                                Class,
                                Type == RISC ? "RISC" : Type == VLIW ? "VLIW" : Type == RED ? "Red" : "nCUBE",
                                Units, Units1, Units2,
                                "Display");
                        send(s, buf, strlen(buf), 0);
                }

                // create a string with a URL representing this query
                char *OutURL() {
                        static char dope[20][1000];
                        static int next = 0;

                        sprintf(dope[next%20],
"/planner?pt=%s&a1=%s&a2=%s&b1=%s&b2=%s&c1=%s&c2=%s&d1=%s&d2=%s&r1=%s&r2=%s&y=%.0lf&y1=%.0lf&y2=%.0lf&class=%d&arch=%s&u=%d&u1=%d&u2=%d&cmd=%s"
,
                                GreekSuffix(PerformanceTarget),
                                GreekSuffix(IO_BytesPerSecondPerFlop1), GreekSuffix(IO_BytesPerSecondPerFlop2),
                                GreekSuffix(IO_BytesPerSecondPerFlopAll1), GreekSuffix(IO_BytesPerSecondPerFlopAll2),
                                GreekSuffix(RAM_BytesPerFlop1), GreekSuffix(RAM_BytesPerFlop2),
                                GreekSuffix(RAM_BytesPerSecondPerFlop1), GreekSuffix(RAM_BytesPerSecondPerFlop2),
                                GreekSuffix(Realization1), GreekSuffix(Realization2),
                                Year, Year1, Year2,
                                Class,
                                Type == RISC ? "RISC" : Type == VLIW ? "VLIW" : Type == RED ? "Red" : "nCUBE",
                                Units, Units1, Units2,
                                "Display");
                        return dope[next++%20];
                }

                // output an HTML form to represent this query and permit editing
                void PrintQuery(FILE *out) {
                        fprintf(out, "%s<form method=get action=planner><table><tr><td valign=top><table border>\n", Out1);
                        fprintf(out, "<tr><th>%sParameter%s</th><th>%sValue%s</th><th>%s %s</th><th>%sUnits%s</th></tr>\n", Out1, Out2,
Out1, Out2, Out1, Out2, Out1, Out2);
                        fprintf(out, "<tr><td>%sPerformance target%s</td><td>%s<input type=text size=8 name=pt
value=\"%s\">%s</td><td> </td><td>%sflops%s</td></tr>\n", Out1, Out2, Out1, GreekSuffix(PerformanceTarget), Out2, Out1, Out2);
                        fprintf(out, "<tr><th> </th><th>%sData Intensive%s</th><th>%sFLOPS Intensive%s</th><th> </th></tr>\n", Out1,
Out2, Out1, Out2);
                        fprintf(out, "<tr><td>%sIO bandwidth%s</td><td>%s<input type=text size=8 name=a1 value=\"%s\">%s</td><td>%s<input
type=text size=8 name=a2 value=\"%s\">%s</td><td>%sbytes/second/flop%s</td></tr>\n", Out1, Out2, Out1, GreekSuffix(IO_BytesPerSecondPerFlop1),
Out2, Out1, GreekSuffix(IO_BytesPerSecondPerFlop2), Out2, Out1, Out2);
                        fprintf(out, "<tr><td>%sIO bandwidth%s</td><td>%s<input type=text size=8 name=b1 value=\"%s\">%s</td><td>%s<input
type=text size=8 name=b2 value=\"%s\">%s</td><td>%sbytes/second/flop all links%s</td></tr>\n", Out1, Out2, Out1,
GreekSuffix(IO_BytesPerSecondPerFlopAll1), Out2, Out1, GreekSuffix(IO_BytesPerSecondPerFlopAll2), Out2, Out1, Out2);
                        fprintf(out, "<tr><td>%sMemory size%s</td><td>%s<input type=text size=8 name=c1 value=\"%s\">%s</td><td>%s<input
type=text size=8 name=c2 value=\"%s\">%s</td><td>%sbytes/flop%s</td></tr>\n", Out1, Out2, Out1, GreekSuffix(RAM_BytesPerFlop1), Out2, Out1,
GreekSuffix(RAM_BytesPerFlop2), Out2, Out1, Out2);
                        fprintf(out, "<tr><td>%sMemory bandwidth%s</td><td>%s<input type=text size=8 name=d1 value=\"%s\">%s</td><td>%s<input
type=text size=8 name=d2 value=\"%s\">%s</td><td>%sbytes/second/flop%s</td></tr>\n", Out1, Out2, Out1, GreekSuffix(RAM_BytesPerSecondPerFlop1),
Out2, Out1, GreekSuffix(RAM_BytesPerSecondPerFlop2), Out2, Out1, Out2);
```

```
                        fprintf(out, "<tr><td>%sRealization factor%s</td><td>%s<input type=text size=8 name=r1 value=\"%s\">%s</td><td>%s<input
type=text size=8 name=r2 value=\"%s\">%s</td><td>%sfraction%s</td></tr>\n", Out1, Out2, Out1, GreekSuffix(Realization1), Out2, Out1,
GreekSuffix(Realization2), Out2, Out1, Out2);

                        fprintf(out, "<tr><th> </th><th>%sFrom%s</th><th>%sTo%s</th><th> </th></tr>\n", Out1, Out2, Out1, Out2);
                        fprintf(out, "<tr><td>%sPeriod%s</td><td>%s<input type=text size=8 name=y1 value=\"%d\">%s</td><td>%s<input type=text
size=8 name=y2 value=\"%d\">%s</td><td>%syear%s</td></tr>\n", Out1, Out2, Out1, (int)Year1, Out2, Out1, (int)Year2, Out2, Out1, Out2);
                        fprintf(out, "<tr><td>%sUnits%s</td><td>%s<input type=text size=8 name=u1 value=\"%d\">%s</td><td>%s<input type=text
size=8 name=u2 value=\"%d\">%s</td><td>%sbytes/second/flop%s</td></tr>\n", Out1, Out2, Out1, Units1, Out2, Out1, Units2, Out2, Out1, Out2);
                        fprintf(out, "<tr><td colspan=4 align=center><input type=submit name=cmd value=\"Calculate
Range\"></td></tr></table></td>\n");

                        if (0) {

                                fprintf(out, "<td valign=top><table border=1>\n");
                                fprintf(out, "<tr><th>%sParameter%s</th><th>%sValue%s</th></tr>\n", Out1, Out2, Out1, Out2);
                                fprintf(out, "<tr><td>%sClass:%s</td><td>%s<input type=radio name=class value=0%s>Data Intensive<br><input
type=radio name=class value=1%s>FLOPS Intensive%s</td></tr>\n",
                                        Out1, Out2, Out1,
                                        Class == 0 ? " checked" : "",
                                        Class != 0 ? " checked" : "",
                                        Out2);

                                fprintf(out, "<tr><td>%sArchitecture:%s</td><td>%s<input type=radio name=arch value=RISC%s> RISC
PIM<br><input type=radio name=arch value=VLIW%s> VLIW PIM<br><input type=radio name=arch value=Red%s> Red<br><input type=radio name=arch
value=nCUBE%s> nCUBE%s</td></tr>\n",
                                                Out1, Out2, Out1,
                                                Type == RISC ? " checked" : "",
                                                Type == VLIW ? " checked" : "",
                                                Type == RED ? " checked" : "",
                                                Type == NCUBE ? " checked" : "",
                                                Out2);

                                fprintf(out, "<tr><td>%sYear:%s</td><td>%s<input type=text size=8 name=y value=\"%d\">%s</td></tr>\n",
Out1, Out2, Out1, (int)Year, Out2);
                                fprintf(out, "<tr><td>%sUnits:%s</td><td>%s<input type=text size=8 name=u value=\"%d\">%s</td></tr>\n",
Out1, Out2, Out1, Units, Out2);
                                fprintf(out, "<tr><td colspan=2 align=center><input type=submit name=cmd
value=Display></td></tr></table></td>\n");
                        }

                        fprintf(out, "</tr></table></form>%s<br>\n", Out2);
                }

};

struct Curve {
                double data[10];
                char *name;
                char *units;
} ITRS2001[][10] = {
                {                                                                                                       // table 0
                        { { 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2010, 2013, 2016, }, "year of production", },
                        { { 130, 115, 100, 90, 80, 70, 65, 45, 32, 22, }, "DRAM 1/2 Pitch", "nm", },
                        { { 150, 130, 107, 90, 80, 70, 65, 45, 32, 22, }, "MPU/ASIC 1/2 Pitch", "nm", },
                        { { 90, 75, 65, 53, 45, 40, 35, 25, 18, 13, }, "MPU Printed Gate Length", "nm", },
                        { { 65, 53, 45, 37, 32, 28, 25, 18, 13, 9, }, "MPU Physical Gate Length", "nm", },
                },
                {                                                                                                       // table 1
                        { { 0, }, "", },
                },
                {                                                                                                       // table 2
                        { { 0, }, "", },
                },
                {                                                                                                       // table 3
                        { { 0, }, "", },
                },
                {                                                                                                       // table 4
                        { { 1684, 2317, 3088, 3990, 5173, 5631, 6739, 11511, 19348, 28751, }, "On-chip Local Clock", "MHz", },
                        { { 1684, 2317, 3088, 3990, 5173, 5631, 6739, 11511, 19348, 28751, }, "Chip-to-board (off-chip) speed", "MHz", },
                },
                {                                                                                                       // table 5
                        { { 0, }, "", },
                },
                {                                                                                                       // table 6
                        { { 1.1, 1.0, 1.0, 1, .9, .9,  7, .6, .5, .4, }, "Vdd (high performance)", "v", },
                },
};

class RoadmapBase {
public:
                double Year;

                double Interpolate(double *vec) {
#if YEAR == 2000
                        static double RoadmapYears[10] = { 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2008, 2011, 2014, };
#else
                        static double RoadmapYears[10] = { 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2010, 2013, 2016, };
#endif
                        if (Year <= RoadmapYears[0]) return vec[0];

                        for (int i = 0; i < 9; i++)
                                if (Year >= RoadmapYears[i] && Year < RoadmapYears[i+1]) {
                                        double frac = (Year-RoadmapYears[i]) / (RoadmapYears[i+1]-RoadmapYears[i]);
                                        return vec[i] * (1.0-frac) + vec[i+1] * frac;
                                }
                        return vec[9];
                }

                double I(Curve *c, double Y) {
                        if (Y <= Dat(0, 0, 0)) return c->data[0];

                        for (int i = 0; i < 9; i++)
                                if (Y >= Dat(0, 0, i) && Y < Dat(0, 0, i+1)) {
                                        double frac = (Y - Dat(0, 0, i)) / (Dat(0, 0, i + 1) - Dat(0, 0, i));
                                        return c->data[i] * (1.0-frac) + c->data[i + 1] * frac;
                                }
                        return c->data[9];
                }

                Curve *D(int Table, int Index) {
                        return &ITRS2001[Table][Index];
                }
```

```
            Curve *DByName(char *key) {
                        for (int i = 0; i < 10; i++)
                                    for (int j = 0; j < 10; j++) {
                                                Curve *t = &ITRS2001[i][j];
                                                if (t->name == NULL) break;
                                                if (strstr(t->name, key) != NULL)
                                                            return t;
                                    }
                        return NULL;
            }

            double Dat(int Table, int Index, int Time) {
                        return D(Table, Index)->data[Time];
            }

/*
            double ITRSAtYear(int Table, int Index, double Y) {
                        if (Y <= Dat(0, 0, 0)) return Dat(Table, Index, 0);

                        for (int i = 0; i < 9; i++)
                                    if (Y >= Dat(0, 0, i) && Y < Dat(0, 0, i+1)) {
                                                double frac = (Y-Dat(0, 0, i)) / (Dat(0, 0, i + 1)-Dat(0, 0, i));
                                                return Dat(Table, Index, i) * (1.0-frac) + Dat(Table, Index, i + 1) * frac;
                                    }
                        return (Table, Index, 9);
            }
*/
            double ITRS(char *key, double Y) {
                        Curve *c = DByName(key);
                        if (c == NULL) return 0.0;
                        return I(c, Y);
            }

            double ITRS(char *key) {
                        return ITRS(key, Year);
            }

};

class Technology: public RoadmapBase {
public:
            virtual double MaxFunctionsPerChip() = 0;       // transistors (logic) or bits (DRAM)
            virtual double MaxChipSize() = 0;                           // square millimeters
            virtual double LocalClock() = 0;                            // intra-chip clock
            virtual double ChipToBoardFast() = 0;                       // speed to PC board
            virtual double Pads() = 0;                                                      // I/O pads
            virtual double HighPerformanceCoreSize() { return 0.0; }
            virtual double GBitsCm() = 0;                                           // if memory -> gigabits per square centimeter
            virtual double CellArea() = 0;                                          // if memory -> square microns per bit
            virtual double Cost() = 0;                                                      // cost per functional unit
            virtual char *Name() = 0;
};

class ASIC_Class: public Technology {
public:
            double MaxFunctionsPerChip() {      // million transistors
#if YEAR == 2000
                        static double data[10] = { 157, 223, 315, 426, 411, 556, 751, 1852, 4568, 11269, };
#else
                        static double data[10] = { 714, 899, 810, 1020, 1286, 1620, 2041, 4081, 8163, 16326, }; // 2002 1i & 1j
#endif
                        return Interpolate(data);
            }

            double MaxChipSize() {  // square millimeters
#define DIESQMM 1600
#define ASICCOST 2000
#if YEAR == 2000
                        static double data[10] = { 800, 800, 800, 800, 572, 572, 572, 572, 572, 572, };
#else
                        static double data[10] = { DIESQMM, DIESQMM, DIESQMM, DIESQMM, DIESQMM, DIESQMM, DIESQMM, DIESQMM, DIESQMM, DIESQMM, };
// 2002 1i & 1j
#endif
                        return Interpolate(data);
            }
            // WRONG!!
            double LocalClock() {               // megahertz
#if YEAR == 2000
                        static double data[10] = { 1250, 1620, 2100, 2490, 2952, 3500, 4150, 7115, 11050, 14920, };
#else
                        static double data[10] = { 1684, 2317, 3088, 3990, 5173, 5631, 6739, 11511, 19348, 28751, }; // 2002 4c & 4d
#endif
                        return Interpolate(data);
            }
            double ChipToBoardFast() {          // megahertz
#if YEAR == 2000
                        static double data[10] = { 1200, 1386, 1600, 1724, 1857, 2000, 2155, 2655, 3190, 3825, };
#else
                        static double data[10] = { 1684, 2317, 3088, 3990, 5173, 5631, 6739, 11511, 19348, 28751, }; // 2002 4c & 4d
#endif
                        return Interpolate(data);
            }
            double Pads() {                                 // pads
#if YEAR == 2000
                        static double data[10] = { 700, 900, 1100, 1300, 1500, 1700, 1900, 2300, 2700, 3000, };
#else
                        static double data[10] = { 1500, 1600, 1700, 1800, 2000, 2100, 2200, 2400, 2700, 3000, }; // 2002 3a & 3b
#endif
                        return Interpolate(data);
            }
            double GBitsCm() { return 0.0; }
            double CellArea() { return 0.0; }
            // No referencable data
            // $1000.00 per ASIC
            double Cost() {                     // packaged microcents per transistor
                        return ASICCOST * 100. / MaxFunctionsPerChip();
            }
            char *Name() { return "ASIC"; }
};

class HighPerformanceMPU_Class: public Technology {
public:
            double MaxFunctionsPerChip() {      // million transistors
#if YEAR == 2000
```

```
                        static double data[10] = { 61, 86, 122, 173, 244, 345, 488, 1381, 3907, 11052, };
#else
                        static double data[10] = { 276, 348, 439, 553, 697, 878, 1106, 2212, 4424, 8848, }; // 2002 1i & 1j
#endif
                        return Interpolate(data);
            }

            double MaxChipSize() {  // million transistors
#if YEAR == 2000
                        static double data[10] = { 310, 310, 310, 325, 340, 356, 372, 427, 489, 561, };
#else
                        static double data[10] = { 310, 310, 310, 310, 310, 310, 310, 310, 310, 310, }; // 2002 1i & 1j
#endif
                        return Interpolate(data);
            }
            double LocalClock() {               // megahertz
#if YEAR == 2000
                        static double data[10] = { 1250, 1620, 2100, 2490, 2952, 3500, 4150, 7115, 11050, 14920, };
#else
                        static double data[10] = { 1684, 2317, 3088, 3990, 5173, 5631, 6739, 11511, 19348, 28751, }; // 2002 4c & 4d
#endif
                        return Interpolate(data);
            }
            double ChipToBoardFast() {          // megahertz
#if YEAR == 2000
                        static double data[10] = { 1200, 1386, 1600, 1724, 1857, 2000, 2155, 2655, 3190, 3825, };
#else
                        static double data[10] = { 1684, 2317, 3088, 3990, 5173, 5631, 6739, 11511, 19348, 28751, }; // 2002 4c & 4d
#endif
                        return Interpolate(data);
            }
            // WRONG!!
            double Pads() {                                 // pads
#if YEAR == 2000
                        static double data[10] = { 700, 900, 1100, 1300, 1500, 1700, 1900, 2300, 2700, 3000, };
#else
                        static double data[10] = { 1500, 1600, 1700, 1800, 2000, 2100, 2200, 2400, 2700, 3000, }; // 2002 3a & 3b
#endif
                        return Interpolate(data);
            }
            double HighPerformanceCoreSize() {  // million transistors
#if YEAR == 2000
                        static double data[10] = { 12, 17, 24, 33, 47, 65, 94, 265, 700, 2100, };           // derived from Kahng 18 July 2001
"work in progress" with Erik's adjustments from HighPerformanceMPUFunctionsPerChip
#else
                        static double data[10] = { 24, 33, 47, 65, 100, 150, 200, 600, 1500, 3200, };       // (extended by Erik) derived from
Kahng 18 July 2001 "work in progress" with Erik's adjustments from HighPerformanceMPUFunctionsPerChip
#endif
                        return Interpolate(data);
            }
            double GBitsCm() { return 0.0; }
            double CellArea() { return 0.0; }
            double Cost() {                     // packaged microcents per transistor
#if YEAR == 2000
                        static double data[10] = { 245, 174/**/, 123, 86/**/, 61, 43/**/, 31, 11/**/, 3.8, 1.35/**/, };
#else
                        static double data[10] = { 97, 69, 49, 34, 24, 17, 12, 4.31, 1.52, .54, }; // 2002 7a & 7b
#endif
                        return Interpolate(data);
            }
            char *Name() { return "High Performance MPU"; }
};

class DRAMProductionClass: public Technology {
public:
            double MaxFunctionsPerChip() { return MaxChipSize() / CellArea(); }
            double MaxChipSize() { '// square millimeters
#if YEAR == 2000
                        static double data[10] = { 131, 129, 127, 141, 157, 175, 147, 205, 191, 268, };
#else
                        static double data[10] = { 127, 100, 118, 93, 147, 116, 183, 181, 239, 238, }; // 2002 1c & 1d
#endif
                        return Interpolate(data);
            }
            double LocalClock() { return 0.0; }
            double ChipToBoardFast() { return 0.0; }
            double Pads() { return 256; }

            double GBitsCm() {      // gigabits/square centimeter
#if YEAR == 2000
                        static double data[10] = { .2, .29, .42, .54, .68, .87, 1.46, 2.97, 8.99, 18.1, };
#else
                        static double data[10] = { .42, .54, .91, 1.15, 1.46, 1.85, 2.35, 4.75, 14.25, 28.85, }; // 2002 1c & 1d
#endif
                        return Interpolate(data);
            }
            double CellArea() {     // square microns
#if YEAR == 2000
                        static double data[10] = { .26, .18, .13, .1, .082, .065, .039, .019, .0064, .0032, };
#else
                        static double data[10] = { .13, .103, .061, .049, .039, .031, .024, .012, .004, .002, }; // 2002 1c & 1d
#endif
                        return Interpolate(data);
            }
            double Cost() {                     // packaged microcents per bit
#if YEAR == 2000
                        static double data[10] = { 15, 10.5/**/, 7.6, 5.3/**/, 3.8, 2.6/**/, 1.9, .675/**/, .24, .085/**/, };
#else
                        static double data[10] = { 7.7, 5.4, 3.8, 2.7, 1.9, 1.4, .96, .34, .12, .042, }; // 2002 7a & 7b
#endif
                        return Interpolate(data);
            }
            char *Name() { return "DRAM Production"; }
};

class DRAMIntroductionClass: public Technology {
public:
            double MaxFunctionsPerChip() { return MaxChipSize() / CellArea(); }
            double MaxChipSize() {  // square millimeters
#if YEAR == 2000
                        static double data[10] = { 400, 395, 390, 435, 485, 542, 454, 516, 392, 448, };
#else
                        static double data[10] = { 390, 308, 364, 287, 454, 359, 568, 563, 373, 186, }; // 2002 1e & 1f
#endif
```

```
                return Interpolate(data);
        }
        double LocalClock() { return 0.0; }
        double ChipToBoardFast() { return 0.0; }
        double Pads() { return 128; }

        double GBitsCm() {      // gigabits/square centimeter
#if YEAR == 2000
                static double data[10] = { .27, .38, .55, .70, .88, 1.12, 1.89, 3.82, 11.56, 23.25, };
#else
                static double data[10] = { .55, .7, 1.18, 1.49, 1.89, 2.39, 3.03, 6.1, 18.42, 37., }; // 2002 1e & 1f
#endif
                return Interpolate(data);
        }
        double CellArea() {     // square microns
#if YEAR == 2000
                static double data[10] = { .26, .18, .13, .1, .082, .065, .039, .019, .0064, .0032, };
#else
                static double data[10] = { .13, .103, .061, .049, .039, .031, .024, .012, .004, .002, }; // 2002 1e & 1f
#endif
                return Interpolate(data);
        }
        double Cost() {         // packaged microcents per bit
#if YEAR == 2000
                static double data[10] = { 42, 30/**/, 21, 15/**/, 11, 7.5/**/, 5.3, 1.8/**/, .66, .23/**/, };
#else
                static double data[10] = { 21, 14.8, 10.5, 7.4, 5.3, 3.7, 2.6, .93, .33, .12, }; // 2002 1e & 1f
#endif
                return Interpolate(data);
        }
        char *Name() { return "DRAM Introduction"; }
};

// implement the SIA Roadmap for Semiconductors
class Roadmap: public RoadmapBase {
public:
        ASIC_Class ASIC;
        HighPerformanceMPU_Class HighPerformanceMPU;
        DRAMIntroductionClass DRAMIntroduction;
        DRAMProductionClass DRAMProduction;

        Roadmap(double y) {
                Year = y;
                ASIC.Year = y;
                HighPerformanceMPU.Year = y;
                DRAMIntroduction.Year = y;
                DRAMProduction.Year = y;
        }
#if 0
        double DRAMHalfPitch() {                                // nanometers
                static double dataDRAMHalfPitch[10] = { 180, 150, 130, 115, 100, 90, 80, 60, 40, 30, };
                return Interpolate(dataDRAMHalfPitch);
        }
        double Roadmap::DRAMCellArea() {    // square microns
                static double dataDRAMCellArea[10] = { .26, .18, .13, .1, .082, .065, .039, .019, .0064, .0032, };
                return Interpolate(dataDRAMCellArea);
        }
        double HighPerformanceMPUCoreSize() {           // million transistors
                static double dataHighPerformanceMPUCoreSize[10] = { 12, 17, 24, 33, 47, 65, 94, 265, 700, 2100, };         // derived
from Kahng 18 July 2001 "work in progress" with Erik's adjustments from HighPerformanceMPUFunctionsPerChip
                return Interpolate(dataHighPerformanceMPUCoreSize);
        }


        double Roadmap::LocalClockAcrossChip() {                // megahertz
                static double dataLocalClock[10] = { 1200, 1386, 1600, 1724, 1857, 2000, 2155, 2655, 3190, 3825, };
                return Interpolate(dataLocalClock);
        }

        double Roadmap::ASICClock() {               // megahertz
                static double dataASICClock[10] = { 500, 559, 626, 700, 761, 828, 900, 1200, 1500, 1800, };
                return Interpolate(dataASICClock);
        }

        double Roadmap::ChipToBoard() {             // megahertz
                static double dataChipToBoard[10] = { 480, 589, 722, 885, 932, 982, 1035, 1285, 1540, 1800, };
                return Interpolate(dataChipToBoard);
        }
#endif
};

class Chip {
        Technology *Tech;
public:
        char *Name;
        int Number;

        double SqMM;

        int CPUCores;
        double FLOPS;
        double ClkMHz;

        double BytesDRAM;

        int IOPads3d;
        double IOBandwidth3d;

        int MemPads;
        double BusBandwidth;

        double CostPerChip;

        Chip();
        void SetTech(Technology *t) { Tech = t; SqMM = Tech->MaxChipSize(); CostPerChip = Tech->MaxFunctionsPerChip() * Tech->Cost() /
100.; }
//      double HighPerformanceCoreSize() { return Tech != NULL ?  Tech->HighPerformanceCoreSize() : 0.0; }
        double Power() {
                double S = Tech->ITRS("MPU/ASIC 1/2 Pitch") / Tech->ITRS("MPU/ASIC 1/2 Pitch", 1990);
                double Vdd = Tech->ITRS("Vdd (high performance)");
                double ClkRatio = Tech->ITRS("On-chip Local Clock") / Tech->ITRS("On-chip Local Clock", 1990);
                double Pwr = S * Vdd * Vdd / Tech->ITRS("Vdd (high performance)", 1990) / Tech->ITRS("Vdd (high performance)", 1990);

                double Pwr200X;
```

```
                        double BaseCPU = 100, BaseMem = 1, BaseRouter = 2;
                        if (CPUCores > 0 && BytesDRAM > 0) {
                                    Pwr200X = BaseMem * Pwr * ClkRatio / S / S;
                                    Pwr200X = CPUCores * BaseCPU * Pwr * ClkRatio;
                        }
                        else if (CPUCores > 0)
                                    Pwr200X = BaseCPU * Pwr * ClkRatio / S / S;
                        else if (BytesDRAM > 0)
                                    Pwr200X = BaseMem * Pwr * ClkRatio / S / S;
                        else
                                    Pwr200X = BaseRouter * Pwr * ClkRatio / S / S;
                        return Pwr200X;
            }
            void Print(FILE *);
};

Chip::Chip() {
            Tech = NULL;
            Name = NULL;
            Number = 0;

            SqMM = 0.0;

            CPUCores = 0;
            FLOPS = 0.0;
            ClkMHz = 0.0;

            BytesDRAM = 0.0;

            IOPads3d = 0;
            IOBandwidth3d = 0.0;

            MemPads = 0;
            BusBandwidth = 0.0;
}

void Chip::Print(FILE *out) {
            if (Number == 0) return;
            fprintf(out, "<h2>%s</h2>\n", Name);
            fprintf(out, "Implemented with %s technology<br>\n", Tech->Name());

            double S = Tech->ITRS("MPU/ASIC 1/2 Pitch") / Tech->ITRS("MPU/ASIC 1/2 Pitch", 1990);
            double Vdd = Tech->ITRS("Vdd (high performance)");
            double ClkRatio = Tech->ITRS("On-chip Local Clock") / Tech->ITRS("On-chip Local Clock", 1990);
            double Pwr = S * Vdd * Vdd / Tech->ITRS("Vdd (high performance)", 1990) / Tech->ITRS("Vdd (high performance)", 1990);

            double Pwr200X;
            double BaseCPU = 100, BaseMem = 1, BaseRouter = 2;
            if (CPUCores > 0 && BytesDRAM > 0) {
                        Pwr200X = BaseMem * Pwr * ClkRatio / S / S;
                        Pwr200X = CPUCores * BaseCPU * Pwr * ClkRatio;
            }
            else if (CPUCores > 0)
                        Pwr200X = BaseCPU * Pwr * ClkRatio / S / S;
            else if (BytesDRAM > 0)
                        Pwr200X = BaseMem * Pwr * ClkRatio / S / S;
            else
                        Pwr200X = BaseRouter * Pwr * ClkRatio / S / S;

            fprintf(out, "1/2 pitch %.0lf->%.0lf S=%.2lf\n", Tech->ITRS("MPU/ASIC 1/2 Pitch", 1990), Tech->ITRS("MPU/ASIC 1/2 Pitch"), S);
            fprintf(out, " Vdd = %.2lf ClkRatio = %.0lf Power=%s fixed %s grow<br>\n", Vdd, ClkRatio, GreekSuffix(Pwr * ClkRatio),
GreekSuffix(Pwr * ClkRatio / S / S));
            fprintf(out, "%sW<br>\n", GreekSuffix(Pwr200X));

            if (Number == 1) fprintf(out, "$%.2lf<br>\n", CostPerChip);
            else fprintf(out, "$%.2lf (%d repetitions of this chip @ $%.2lf)<br>\n", CostPerChip * Number, Number, CostPerChip);

            if (SqMM != 0.0) fprintf(out, "%.1lf Sq MM<br>\n", SqMM);

            if (CPUCores != 0) fprintf(out, "%d CPU Cores<br>\n", CPUCores);
            if (FLOPS > 1e8) fprintf(out, "%.1lf GFLOPS<br>\n", FLOPS/1e9);
            else if (FLOPS != 0.0) fprintf(out, "%.1lf MFLOPS<br>\n", FLOPS/1e6);

            if (ClkMHz > 1e2) fprintf(out, "%.1lf GHz Clock<br>\n", ClkMHz/1e3);
            else if (ClkMHz != 0.0) fprintf(out, "%.1lf MHz Clock<br>\n", ClkMHz);

            if (BytesDRAM * Number > 2e9) fprintf(out, "%.1lf GBytes DRAM (%d repetitions of chip with %.1lf MBits)<br>\n", Number *
BytesDRAM/1024./1024./1024., Number, BytesDRAM/1024./1024.*8);
            else if (BytesDRAM != 0.0) fprintf(out, "%.1lf MBytes DRAM (%d repetitions of chip with %.1lf MBits)<br>\n", Number *
BytesDRAM/1024./1024., Number, BytesDRAM/1024./1024.*8);

            if (IOBandwidth3d != 0) fprintf(out, "%sBytes/s x 6 I/O bandwidth on %d x 6 pins<br>\n", GreekSuffix(IOBandwidth3d), IOPads3d);

            if (BusBandwidth != 0) fprintf(out, "%sBytes/s bus bandwidth on %d pins<br>\n", GreekSuffix(BusBandwidth), MemPads);

            double PinScale = .075;

            int TotalPins = (int)(Tech->Pads() * PinScale);
            int BusPins = (int)(MemPads * PinScale);
            if (BusPins == 0) BusPins++;
            int IOPins = (int)(IOPads3d * PinScale);
            int UnusedPins = TotalPins -BusPins - 6*IOPins;


            fprintf(out, "<table border=1 cellspacing=0 cellpadding=0 align=center><tr><th>%s%d Pins Max%s</th></tr><tr><td>\n", Index1,
(int)Tech->Pads(), Index2);
            fprintf(out, "<table border=0 cellspacing=0 cellpadding=0>\n");
            for (int i = 0, e = Number > 1 ? Number : 1; i < e; i++) {
                        if (i != 0)
                                    fprintf(out, "<tr></tr>\n");
                        fprintf(out, "<tr>\n");
                        if (IOPins > 0)
                                    for (int j = 0; j < 6; j++)
                                                fprintf(out, "<td><img src=%s.gif width=%d height=10></td>\n", j%2 ? "blue" : "red", IOPins);
                        fprintf(out, "<td><img src=yellow.gif width=%d height=10></td>\n", BusPins);
                        if (UnusedPins > 0)
                                    fprintf(out, "<td><img src=green.gif width=%d height=10></td>\n", UnusedPins);
                        fprintf(out, "</tr>");
            }
            fprintf(out, "</table>\n");
            fprintf(out, "</td></tr>");
            fprintf(out, "<tr><td>%sRed/Blue = I/O Link<br>Yellow = Memory Bus<br>Green = Available%s</td></tr>\n", Index1, Index2);
            fprintf(out, "</table>\n");
}
```

```
class Picture {
public:
            FILE *out;
            int CPUChips;
            int RamChips;
            double CPUSz;
            double DRAMSz;
            char *CPUText;
            char *DRAMText;
            char *PadText;

            void Set(FILE *o, int c, int r, double cz, double rz, char *ct, char *rt, char *pt) { out = o; CPUChips = c; RamChips = r; CPUSz =
cz; DRAMSz = rz; CPUText = ct; DRAMText = rt; PadText = pt; }
            virtual void Print() = 0;
};

// output HTML for an ASCI Red style node with a CPU and RAM chips
class nCUBEPicture: public Picture {
public:
            void Print();
};

void nCUBEPicture::Print() {
            int RAMArray;
            for (RAMArray = 1; RAMArray*RAMArray < RamChips; RAMArray++) ;

            fprintf(out, "<table border=1 cellspacing=0 cellpadding=0 bgcolor=%s align=center><tr><td>\n", PCBColor);
            fprintf(out, "<table border=0 cellspacing=%d cellpadding=%d bgcolor=%s><tr><td valign=center>\n", CS, CP, PCBColor);

            int CPUEdge = (int)(SCALE * sqrt(CPUSz));
            fprintf(out, "<table border=0 cellspacing=%d cellpadding=%d>\n", CS, CP);
            fprintf(out, "<tr><td><img src=%s width=1 height=1></td><td><img src=%s width=%d height=1></td><td><img src=%s width=1
height=1></td></tr>\n", PCBoardGIF, PCBoardGIF, CPUEdge, PCBoardGIF);
            fprintf(out, "<tr><td><img src=%s width=1 height=%d></td><td align=center bgcolor=%s><font color=white>%s%s%s</font></td><td><img
src=%s width=1 height=%d></td></tr>\n", PCBoardGIF, CPUEdge, CPUColorA, Chip1, "CPU<br>+<br>Router", Chip2, PCBoardGIF, CPUEdge);
            fprintf(out, "<tr><td><img src=%s width=1 height=1></td><td><img src=%s width=%d height=1></td><td><img src=%s width=1
height=1></td></tr>\n", PCBoardGIF, PCBoardGIF, CPUEdge, PCBoardGIF);
            fprintf(out, "</table>\n");

            fprintf(out, "</td><td valign=center>\n");

            fprintf(out, "<table border=0 cellspacing=%d cellpadding=%d>\n", CS, CP);
            int DRAMEdge = (int)(SCALE * sqrt(DRAMSz));

            fprintf(out, "<tr>\n");
            if (1) for (int j = 0; j < RAMArray; j++)
                        fprintf(out, "<td><img src=%s height=1 width=1></td><td><img src=%s height=1 width=%d></td><td><img src=%s height=1
width=1></td>\n", PCBoardGIF, PCBoardGIF, DRAMEdge, PCBoardGIF);
            fprintf(out, "</tr>\n");

            for (int i = 0; i < RAMArray; i++) {
                        fprintf(out, "<tr>\n");
                        for (int j = 0; j < RAMArray; j++)
                                    if ((RAMArray-i-1)*RAMArray + j < RamChips)
                                                fprintf(out, "<td><img src=%s height=%d width=1></td><td bgcolor=%s
align=center>%sDRAM%s</td><td><img src=%s height=%d width=1></td>\n", PCBoardGIF, DRAMEdge, (i+j)%2 ? DRAMColor1 : DRAMColor2, Chip1, Chip2,
PCBoardGIF, DRAMEdge);
                        fprintf(out, "<tr>\n");
            }
            fprintf(out, "</table>\n");

            fprintf(out, "</td></tr></table>\n");
            fprintf(out, "</td></tr></table>\n");

}

// output HTML for an ASCI Red style node with a CPU and RAM chips
class RedPicture: public Picture {
public:
            void Print();
};

void RedPicture::Print() {
            fprintf(out, "<table border=1 cellspacing=0 cellpadding=0 bgcolor=%s align=center><tr><td valign=center>\n", PCBColor); // border
            fprintf(out, "<table border=0 cellspacing=%d cellpadding=%d bgcolor=%s><tr><td valign=bottom>\n", CS, CP, PCBColor);    // a/b c

            int CPUArray, aspectratio=3;
            for (CPUArray = 1; aspectratio*CPUArray*CPUArray < CPUChips; CPUArray++) ;

            fprintf(out, "<table border=0 cellspacing=%d cellpadding=%d align=center>\n", CS, CP);          // a
            int CPUEdge = (int)(SCALE * sqrt(DRAMSz));

            fprintf(out, "<tr>\n");
            for (int j = 0; j < CPUArray; j++)
                        fprintf(out, "<td><img src=%s height=1 width=1></td><td><img src=%s height=1 width=%d></td><td><img src=%s height=1
width=1></td>\n", PCBoardGIF, PCBoardGIF, CPUEdge, PCBoardGIF);
            fprintf(out, "</tr>\n");

            if (1) for (int i = 0; i < aspectratio*CPUArray; i++) {
                        fprintf(out, "<tr>\n");
                        for (int j = 0; j < CPUArray; j++)
                                    if ((aspectratio*CPUArray-i-1)*CPUArray + j < CPUChips)
                                                fprintf(out, "<td><img src=%s height=%d width=1></td>"
                                                            "<td bgcolor=%s align=center><font color=white>%sCPU%s</font></td>"
                                                            "<td><img src=%s height=%d width=1></td>\n", PCBoardGIF, CPUEdge,
(i+j)%2 ? CPUColorB : CPUColorA, Chip1, Chip2, PCBoardGIF, CPUEdge);
                        fprintf(out, "<tr>\n");
            }
            fprintf(out, "</table>\n");                                 // a

            fprintf(out, "</td><td rowspan=2 valign=center>\n");        // a/b c

            int RAMArray;
            for (RAMArray = 1; RAMArray*RAMArray < RamChips; RAMArray++) ;

            fprintf(out, "<table border=0 cellspacing=%d cellpadding=%d align=center>\n", CS, CP);          // c
            int DRAMEdge = (int)(SCALE * sqrt(DRAMSz));

            fprintf(out, "<tr>\n");
            if (1) for (int j = 0; j < RAMArray; j++)
                        fprintf(out, "<td><img src=%s height=1 width=1></td><td><img src=%s height=1 width=%d></td><td><img src=%s height=1
width=1></td>\n", PCBoardGIF, PCBoardGIF, DRAMEdge, PCBoardGIF);
            fprintf(out, "</tr>\n");
```

```
            if (1) for (int i = 0; i < RAMArray; i++) {
                        fprintf(out, "<tr>\n");
                        for (int j = 0; j < RAMArray; j++)
                                    if ((RAMArray-i-1)*RAMArray + j < RamChips)
                                                fprintf(out, "<td><img src=%s height=%d width=1></td>"
                                                                                                    "<td bgcolor=%s align=center>%sDRAM%s</td>"
                                                                                                    "<td><img src=%s height=%d width=1></td>\n", PCBoardGIF, DRAMEdge,
(i+j)%2 ? DRAMColor1 : DRAMColor2, Chip1, Chip2, PCBoardGIF, DRAMEdge);
                        fprintf(out, "<tr>\n");
            }
            fprintf(out, "</table>\n");                                                 // c

            fprintf(out, "</td></tr><tr><td valign=top>\n");            // a/b c

            int RouterEdge = (int)(SCALE * sqrt(CPUSz));
            fprintf(out, "<table border=0 cellspacing=0 cellpadding=0 align=left>\n");          // b
            fprintf(out, "<tr><td><img src=%s width=1 height=1></td><td><img src=%s width=%d height=1></td><td><img src=%s width=1
height=1></td></tr>\n", PCBoardGIF, PCBoardGIF, RouterEdge, PCBoardGIF);
            fprintf(out, "<tr><td><img src=%s width=1 height=%d></td><td align=center bgcolor=black><font
color=white>%s%s%s</b></font></td><td><img src=%s width=1 height=%d></td></tr>\n", PCBoardGIF, RouterEdge, Chip1, "Router", Chip2, PCBoardGIF,
RouterEdge);
            fprintf(out, "<tr><td><img src=%s width=1 height=1></td><td><img src=%s width=%d height=1></td><td><img src=%s width=1
height=1></td></tr>\n", PCBoardGIF, PCBoardGIF, RouterEdge, PCBoardGIF);
            fprintf(out, "</table>\n");                                                 // b

            fprintf(out, "</td></tr></table>\n");           // a/b c
            fprintf(out, "</td></tr></table>\n");           // border

}

// output HTML for a PIM type chip
class PIMPicture: public Picture {
public:
            void Print();
};

void PIMPicture::Print() {
            int Units = CPUChips;

            double ChipEdge = sqrt(CPUSz + DRAMSz);

            double CPUEdge = sqrt(CPUSz);
//          double Border = (ChipEdge-CPUEdge)/2;

            int iPad = 2;

//          char *fontsize="size=-12 color=green";

            double scale = SCALE;
            int iEdge = (int)(scale*ChipEdge);
            int iCPUEdge = (int)(scale*CPUEdge);
            int iBorder = (iEdge - iCPUEdge)/2;
            if (iBorder + iBorder + iCPUEdge < iEdge) iCPUEdge++;

            if (out == NULL)
                        return;

            // cut number of units to prevent CPUs less than one pixel
            if (Units > iCPUEdge)
                        Units = iCPUEdge;

            // 1-pixel top line to control spacing
            fprintf(out, "<table border=0 cellspacing=0 cellpadding=0 align=center>\n"
                                    "<tr><td><img src=black.gif width=%d height=%d></td>\n"
                                    "<td><img src=black.gif width=%d height=%d></td>\n", iPad, iPad, iBorder, iPad);
            if (1) for (int i = 0; i < Units; i++) {
                        int left = i * iCPUEdge / Units;
                        int right = (i+1) * iCPUEdge / Units;
                        fprintf(out, "<td><img src=black.gif width=%d height=%d></td>\n", right-left, iPad);
            }
            fprintf(out, "<td><img src=black.gif width=%d height=%d></td>\n"
                                    "<td><img src=black.gif width=%d height=%d></td></tr>\n\n", iBorder, iPad, iPad, iPad);

            // top DRAM region
            char note[100];
            sprintf(note, Units == 1 ? "CPU" : "%d PIM CPUs", Units);
            fprintf(out, "<tr><td><img src=black.gif width=%d height=%d></td>\n"
                                    "<td colspan=%d valign=bottom><img src=yellow.gif width=%d height=%d></td>\n"
                                    "<td><img src=black.gif width=%d height=%d></td></tr>\n\n", iPad, iBorder, Units+2, iEdge, iBorder, iPad,
iBorder);

            // center DRAM-CPU-DRAM region
            fprintf(out, "<tr><td><img src=black.gif width=%d height=%d></td>\n"
                                    "<td><img src=yellow.gif width=%d height=%d></td>\n", iPad, iCPUEdge, iBorder, iCPUEdge);
            if (1) for (int i = 0; i < Units; i++) {
                        int left = i * iCPUEdge / Units;
                        int right = (i+1) * iCPUEdge / Units;
                        fprintf(out, "<td><img src=%s width=%d height=%d></td>\n", (i%2) ? "red.gif" : "blue.gif", right-left, iCPUEdge);
            }
            fprintf(out,"<td><img src=yellow.gif width=%d height=%d></td>\n"
                                    "<td><img src=black.gif width=%d height=%d></td></tr>\n\n", iBorder, iCPUEdge, iPad, iCPUEdge);

            // bottom DRAM region (with CPU legend)

            fprintf(out, "<tr><td><img src=black.gif width=%d height=%d></td>\n"
                                    "<td colspan=%d><img src=yellow.gif width=%d height=%d></td>\n"
                                    "<td><img src=black.gif width=%d height=%d></td></tr>\n\n", iPad, iBorder, 2+Units, iEdge, iBorder, iPad,
iBorder);

            // 1-pixel bottom line to set spacing
            fprintf(out, "<tr><td><img src=black.gif width=%d height=%d></td>\n"
                                    "<td><img src=black.gif width=%d height=%d></td>\n", iPad, iPad, iBorder, iPad);
            if (1) for (int i = 0; i < Units; i++) {

                        int left = i * iCPUEdge / Units;
                        int right = (i+1) * iCPUEdge / Units;
                        fprintf(out, "<td><img src=black.gif width=%d height=%d></td>\n", right-left, iPad);
            }
            fprintf(out, "<td><img src=black.gif width=%d height=%d></td>\n"
                                    "<td><img src=black.gif width=%d height=%d></td></tr>\n</table>\n\n", iBorder, iPad, iPad, iPad);

            return;
}
```

```
void GIFHeader(SOCKET theClient) {
            send(theClient, "HTTP/1.0 200\r\n", 14, 0);
            send(theClient, "Content-type: ", 14, 0);
            send(theClient, "image/gif", 9, 0);
            send(theClient, "\r\n", 2, 0);
            send(theClient, "\r\n", 2, 0);
}

// helper to write a 1x1 pixel gif of a certain color
void SmallGIFHelper(SOCKET theClient, unsigned char *tail) {
            GIFHeader(theClient);
            unsigned char header[73] = { 71, 73, 70, 56, 55, 97, 1, 0, 1, 0, 179, 0, 0, 0, 0, 0, 128, 0, 0, 0, 128, 0, 128, 128, 0, 0, 0, 128,
128, 0, 128, 0, 128, 128, 192, 192, 192, 128, 128, 128, 255, 0, 0, 0, 255, 0, 255, 255, 0, 0, 0, 255, 255, 0, 255, 0, 255, 255, 255, 255, 255,
44, 0, 0, 0, 0, 1, 0, 1, 0, 0, 4, 2, };
            send(theClient, (char *)header, 73, 0);
            send(theClient, (char *)tail, 4, 0);
}

// helper to write a 1x1 pixel gif of a certain color
void BigGIFHelper(SOCKET theClient, unsigned char *data, int len) {
            GIFHeader(theClient);
            send(theClient, (char *)data, len, 0);
}

// Plan an MPP in accordance with the query
// return value indicates an impossible design (error)
// return price per node and units per designated overall performance
int FloorPlanner(FILE *out, Query &qx, double &DollarsPerNode, double &UnitsPerPetaFlop) {

            int rval = 0;                                                                   // error

            double IO_BytesPerSecondPerFlop = qx.Class == 0 ? qx.IO_BytesPerSecondPerFlop1: qx.IO_BytesPerSecondPerFlop2;          // per link
            double IO_BytesPerSecondPerFlopAll = qx.Class == 0 ? qx.IO_BytesPerSecondPerFlopAll1 : qx.IO_BytesPerSecondPerFlopAll2; // all
links together
            double RAM_BytesPerFlop = qx.Class == 0 ? qx.RAM_BytesPerFlop1 : qx.RAM_BytesPerFlop2;                              // memory
size
            double RAM_BytesPerSecondPerFlop = qx.Class == 0 ? qx.RAM_BytesPerSecondPerFlop1 : qx.RAM_BytesPerSecondPerFlop2;      // memory
bandwidth
            double Realization = qx.Class == 0 ? qx.Realization1 : qx.Realization2;

            double MemoryDerator = pow(qx.PerformanceTarget/20e12, -.25);
            RAM_BytesPerFlop *= MemoryDerator;

            Roadmap y(qx.Year);

            // draw navigation portion
            if (out != NULL) {

        fprintf(out, "<html><head><title>%d %s for %d</title></head>\n", qx.Units,
                                    qx.Type == RISC ? "RISC CPU PIM" : qx.Type == VLIW ? "VLIW CPU PIM" : qx.Type == RED ? "CPU Red MPP" : "CPU
Core nCUBE/Blue Light",
                                    (int)qx.Year);
        fprintf(out, "<body background=\"bkgrnd.gif\" text=\"#000000\" link=\"#003366\" vlink=\"#cc0033\" alink=\"#000000\">\n");
        fprintf(out, "<a name=\"TOP\"></a>\n");
        fprintf(out, "<table border=\"0\"  width=\"100%%\">\n");
        fprintf(out, "<tr valign=\"top\"><td>\n");
        fprintf(out, "<table border=\"0\" width=\"140\" valign=\"top\">\n");
        fprintf(out, "<tr valign=\"top\"><td width=\"140\" valign=\"top\">\n");
        fprintf(out, "\n");
//          fprintf(out, "<font color=white>Nav text</font>");

                        fprintf(out, "<table align=center cellspacing=0 cellpadding=0><tr><td><table>\n");
                        fprintf(out, "<tr><td> </td><td align=center colspan=2><a href=/index.htm target=query>%sIndex%s</a><br><a
href=/help.htm>%sHelp%s</a></td><td> </td></tr>\n", NOut1, NOut2, NOut1, NOut2);

                        // navigate years
                        if (qx.Year == qx.Year1)
                                    fprintf(out, "<tr><td align=center>%s---%s</td>", NOut1, NOut2);
                        else {
                                    qx.Year--;
                                    fprintf(out, "<tr><td align=center><a href=%s>%s%d%s</a></td>", qx.OutURL(), NOut1, ((int)qx.Year), NOut2);
                                    qx.Year++;
                        }
                        fprintf(out, "<td align=center colspan=2%s>%s%d%s</td>", NOut1y, NOut1, (int)qx.Year, NOut2);
                        if (qx.Year == qx.Year2)
                                    fprintf(out, "<td align=center>%s---%s</td></tr>\n", NOut1, NOut2);
                        else {
                                    qx.Year++;
                                    fprintf(out, "<td align=center><a href=%s>%s%d%s</a></td></tr>\n", qx.OutURL(), NOut1, (int)qx.Year,
NOut2);
                                    qx.Year--;
                        }

                        // navigate units
                        if (qx.Units == qx.Units1)
                                    fprintf(out, "<tr><td align=center>%s---%s</td>", NOut1, NOut2);
                        else {
                                    qx.Units--;
                                    fprintf(out, "<tr><td align=center><a href=%s>%s%d<br>CPUs%s</a></td>", qx.OutURL(), NOut1, qx.Units,
NOut2);
                                    qx.Units++;
                        }
                        fprintf(out, "<td align=center colspan=2%s>%s%d<br>CPUs%s</td>", NOut1y, NOut1, qx.Units, NOut2);
                        if (qx.Units == qx.Units2)
                                    fprintf(out, "<td align=center>%s---%s</td></tr>\n", NOut1, NOut2);
                        else {
                                    qx.Units++;
                                    fprintf(out, "<td align=center><a href=%s>%s%d<br>CPUs%s</a></td></tr>\n", qx.OutURL(), NOut1, qx.Units,
NOut2);
                                    qx.Units--;
                        }

                        // navigate architectures
                        fprintf(out, "<tr>");
                        CoreType csave = qx.Type;
                        if (csave != RISC) {
                                    qx.Type = RISC;
                                    fprintf(out, "<td align=center><a href=%s>%sRISC%s</a></td>", qx.OutURL(), NOut1, NOut2);
                        }
                        else
                                    fprintf(out, "<td align=center%s>%sRISC%s</td>", NOut1y, NOut1, NOut2);

                        if (csave != VLIW) {
```

```
                    qx.Type = VLIW;
                    fprintf(out, "<td align=center><a href=%s>%sVLIW%s</a></td>", qx.OutURL(), NOut1, NOut2);
                }
                else
                    fprintf(out, "<td align=center%s>%sVLIW%s</td>", NOut1y, NOut1, NOut2);

                if (csave != RED) {
                    qx.Type = RED;
                    fprintf(out, "<td align=center><a href=%s>%sRED%s</a></td>", qx.OutURL(), NOut1, NOut2);
                }
                else
                    fprintf(out, "<td align=center%s>%sRED%s</td>", NOut1y, NOut1, NOut2);

                if (csave != NCUBE) {
                    qx.Type = NCUBE;
                    fprintf(out, "<td align=center><a href=%s>%snCUBE%s</a></td>", qx.OutURL(), NOut1, NOut2);
                }
                else
                    fprintf(out, "<td align=center%s>%snCUBE%s</td>", NOut1y, NOut1, NOut2);
                qx.Type = csave;

                fprintf(out, "</tr>\n");

                fprintf(out, "</td></tr></table></table><p>\n");
    fprintf(out, "</td></tr>\n");
    fprintf(out, "</table>\n");
    fprintf(out, "\n");
    }

    Technology &DRAMTech = y.DRAMProduction;
    double DRAMSizeSqMM = DRAMTech.MaxChipSize();
    double DRAMGBitsCm = DRAMTech.GBitsCm();
    double DRAMCostPerBit = DRAMTech.Cost();
    double BitsDRAM = DRAMSizeSqMM / 100.0 * DRAMGBitsCm * 1024. * 1024. * 1024.;
    double DRAMCost = BitsDRAM * DRAMCostPerBit / 1000000. / 100.;

    Technology &CustomChipTech = y.ASIC;
    double CustomChipSqMM = CustomChipTech.MaxChipSize();
    double CustomChipMTransistorCellSize = CustomChipTech.MaxChipSize() / (CustomChipTech.MaxFunctionsPerChip() * Realization);

    Technology &MPUTech = y.HighPerformanceMPU;
//  double MPUCost = MPUTech.Cost() * MPUTech.MaxFunctionsPerChip() / 100.;

    // these will be the chips
    Chip CPUChip;                       // PIM, Blue Light, or standard CPU
    Chip MemChip;                       // standard DRAM or unused
    Chip RouterChip;        // wormhole router or unused

    Picture *Table;

    // engineering parameters for a PIM with a RISC or VLIW CPU
    if (qx.Type == RISC || qx.Type == VLIW) {
                CPUChip.Name = "PIM";
                CPUChip.SetTech(&CustomChipTech);
                CPUChip.Number = 1;

                CPUChip.ClkMHz = CustomChipTech.LocalClock();
                CPUChip.FLOPS = CPUChip.ClkMHz * qx.Units * (qx.Type == VLIW ? 2 : 1) * 1e6;
                CPUChip.CPUCores = qx.Units;

                CPUChip.IOPads3d = (int)(CustomChipTech.Pads() / 6.);
                CPUChip.IOBandwidth3d = CustomChipTech.ChipToBoardFast() * CPUChip.IOPads3d / 8.0 * 1000000.;

                double CPUSizeSqMM = (qx.Type == VLIW ? 12000000 : 3000000)/1000000.0 * qx.Units * CustomChipMTransistorCellSize;

                // calculate max DRAM area
                double FlopsFromIOBytes = CPUChip.IOBandwidth3d / IO_BytesPerSecondPerFlop;
                double BytesDRAMFromIOBytes = FlopsFromIOBytes * RAM_BytesPerFlop;
                double BitsDRAMFromIOBytes = BytesDRAMFromIOBytes * 8;
                double DRAMSizeSqMMFromIOBytes = BitsDRAMFromIOBytes * DRAMTech.CellArea() / 1000000.;

                double DRAMSizeSqMM = CustomChipSqMM - CPUSizeSqMM;
                if (DRAMSizeSqMM < 0.0) DRAMSizeSqMM = 0.0;
                if (DRAMSizeSqMM > DRAMSizeSqMMFromIOBytes)
                    DRAMSizeSqMM = DRAMSizeSqMMFromIOBytes;

                CPUChip.SqMM = CPUSizeSqMM + DRAMSizeSqMM;      // update this parameter

                double OBitsDRAM = DRAMSizeSqMM * 1000000.0 / DRAMTech.CellArea();
                double BitsDRAM = DRAMSizeSqMM/100*DRAMTech.GBitsCm() * 1e9;
                CPUChip.BytesDRAM = BitsDRAM / 8.0;

                CPUChip.BusBandwidth = CPUChip.FLOPS * RAM_BytesPerSecondPerFlop;

                // derate
                double DeratedFlops = CPUChip.FLOPS;
                if (DeratedFlops > CPUChip.IOBandwidth3d / IO_BytesPerSecondPerFlop)
                    DeratedFlops = CPUChip.IOBandwidth3d / IO_BytesPerSecondPerFlop;
                if (DeratedFlops > CPUChip.BytesDRAM / RAM_BytesPerFlop)
                    DeratedFlops = CPUChip.BytesDRAM / RAM_BytesPerFlop;
                if (DeratedFlops > CPUChip.BusBandwidth / RAM_BytesPerSecondPerFlop)
                    DeratedFlops = CPUChip.BusBandwidth / RAM_BytesPerSecondPerFlop;

                CPUChip.IOBandwidth3d = DeratedFlops * IO_BytesPerSecondPerFlop;
                CPUChip.IOPads3d = 1+(int)(CPUChip.IOBandwidth3d / 1000000. * 8. / CustomChipTech.ChipToBoardFast());

                CPUChip.BytesDRAM = DeratedFlops * RAM_BytesPerFlop;
                DRAMSizeSqMM = CPUChip.BytesDRAM * 8. * DRAMTech.CellArea() / 1000000.;

                DollarsPerNode = CPUChip.CostPerChip;

                Table = new PIMPicture();
                Table->Set(out, qx.Units, 1, CPUSizeSqMM, DRAMSizeSqMM, "cpu", "dram", "pad");
                if ((CPUChip.BytesDRAM + MemChip.BytesDRAM) <= 0.0)
                    rval = 1;

    }

    // engineering parameters for an ASCI Red style node
    else if (qx.Type == RED) {
                CPUChip.Name = "CPU";
                MemChip.Name = "DRAM";
                RouterChip.Name = "Wormhole";
```

```
                        // Divide I/O pins into 6 IPC links plus a memory interface
                        CPUChip.SetTech(&MPUTech);
                        MemChip.SetTech(&DRAMTech);
                        RouterChip.SetTech(&CustomChipTech);
                        CPUChip.Number = qx.Units;
                        CPUChip.ClkMHz = MPUTech.LocalClock() * Realization;

                        // estimate performance for a state-of-the art MPU
                        // assume some number of single issue cores @ 6 million transistors or double issue cores @ 30 million
                        double cs = MPUTech.HighPerformanceCoreSize();
                        CPUChip.FLOPS = MPUTech.LocalClock() * Realization * 1000000.;
                        if (cs > 40) {
                                    CPUChip.CPUCores = (int)cs/30;
                                    CPUChip.FLOPS *= CPUChip.CPUCores * 2;
                        }
                        else {
                                    CPUChip.CPUCores = (int)cs/6;
                                    CPUChip.FLOPS *= CPUChip.CPUCores;
                        }

                        RouterChip.IOPads3d = (int)(CustomChipTech.Pads()/7.);
                        RouterChip.MemPads = (int)(CustomChipTech.Pads() - RouterChip.IOPads3d * 6);
                        RouterChip.IOBandwidth3d = CustomChipTech.ChipToBoardFast() * Realization * RouterChip.IOPads3d * BUSFACTOR / 8.0 *
1000000.;
                        RouterChip.CostPerChip = 500.00;
                        RouterChip.Number = 1;
//                      double MaxFLOPSFromIO = RouterChip.IOBandwidth3d / IO_BytesPerSecondPerFlop;

                        CPUChip.MemPads = (int)MPUTech.Pads();
                        CPUChip.BusBandwidth = CPUChip.MemPads * MPUTech.ChipToBoardFast() * Realization * BUSFACTOR / 8. * 1000000.;

                        MemChip.MemPads = (int)DRAMTech.Pads();
                        MemChip.BusBandwidth = MemChip.MemPads * MPUTech.ChipToBoardFast() * Realization * BUSFACTOR / 8. * 1000000.;
                        MemChip.BytesDRAM = BitsDRAM / 8.0;
//                      double MaxFLOPSFromBus = min(CPUChip.BusBandwidth, MemChip.BusBandwidth) / RAM_BytesPerSecondPerFlop;

                        double DeratedFlops = 0.0;
                        {
                                    double BestCost = 1e15;
                                    for (int t = 0; t < 256; t++) {
                                                DollarsPerNode = CPUChip.CostPerChip * qx.Units + DRAMCost * t + RouterChip.CostPerChip;

                                                double df = CPUChip.FLOPS * qx.Units;
                                                if (df > RouterChip.IOBandwidth3d / IO_BytesPerSecondPerFlop)
                                                            df = RouterChip.IOBandwidth3d / IO_BytesPerSecondPerFlop;
                                                if (df > MemChip.BytesDRAM * t / RAM_BytesPerFlop)
                                                            df = MemChip.BytesDRAM * t / RAM_BytesPerFlop;
                                                if (df > CPUChip.BusBandwidth / RAM_BytesPerSecondPerFlop)
                                                            df = CPUChip.BusBandwidth / RAM_BytesPerSecondPerFlop;
                                                if (df > MemChip.BusBandwidth * t / RAM_BytesPerSecondPerFlop)
                                                            df = MemChip.BusBandwidth * t / RAM_BytesPerSecondPerFlop;
                                                if (df > 0.0 && BestCost > DollarsPerNode / df) {
                                                            MemChip.Number = t;
                                                            BestCost = DollarsPerNode / df;
                                                            DeratedFlops = df;
                                                }
                                    }
                        }

                        DollarsPerNode = CPUChip.CostPerChip * qx.Units + DRAMCost * MemChip.Number + RouterChip.CostPerChip;

                        RouterChip.IOBandwidth3d = DeratedFlops * IO_BytesPerSecondPerFlop;
                        RouterChip.IOPads3d = (int)(RouterChip.IOBandwidth3d / 1000000. * 8. / BUSFACTOR / MPUTech.ChipToBoardFast() /
Realization);

                        Table = new RedPicture();
                        Table->Set(out, qx.Units, MemChip.Number, MPUTech.MaxChipSize(), DRAMSizeSqMM, "cpu", "dram", "pad");
            }

            // engineering parameters for an ASCI Blue Light style node
            else {
                        CPUChip.Name = "ASIC";
                        MemChip.Name = "DRAM";

                        CPUChip.SetTech(&CustomChipTech);
                        MemChip.SetTech(&DRAMTech);
                        CPUChip.Number = 1;
                        CPUChip.ClkMHz = CustomChipTech.LocalClock() * Realization;
                        CPUChip.FLOPS = CPUChip.ClkMHz * qx.Units * 2. * 1000000.;
                        CPUChip.CPUCores = qx.Units;

                        double FracPinsToMemory = RAM_BytesPerSecondPerFlop/(max(IO_BytesPerSecondPerFlop * 6., IO_BytesPerSecondPerFlopAll) +
RAM_BytesPerSecondPerFlop);
                        double FracPinsToIO = 1. - FracPinsToMemory;

                        CPUChip.IOPads3d = (int)(CustomChipTech.Pads()*FracPinsToIO/6. + .5);
                        CPUChip.IOBandwidth3d = CustomChipTech.ChipToBoardFast() * Realization * CPUChip.IOPads3d / 8.0 * 1000000.;
                        CPUChip.MemPads = (int)(CustomChipTech.Pads()*FracPinsToMemory + .5);
                        CPUChip.BusBandwidth = CustomChipTech.Pads() * CustomChipTech.ChipToBoardFast() * Realization * BUSFACTOR *
FracPinsToMemory / 8.0 * 1000000.;

                        MemChip.MemPads = (int)DRAMTech.Pads();
                        MemChip.BusBandwidth = MemChip.MemPads * MPUTech.ChipToBoardFast() * Realization * BUSFACTOR / 8. * 1000000.;
                        MemChip.BytesDRAM = BitsDRAM / 8.0;

                        {
                                    double BestCost = 1e15;
                                    for (int t = 0; t < 256; t++) {
                                                DollarsPerNode = CPUChip.CostPerChip + t * DRAMCost;
                                                double df = CPUChip.FLOPS;
                                                if (df > CPUChip.IOBandwidth3d / IO_BytesPerSecondPerFlop)
                                                            df = CPUChip.IOBandwidth3d / IO_BytesPerSecondPerFlop;
                                                if (df > MemChip.BytesDRAM * t / RAM_BytesPerFlop)
                                                            df = MemChip.BytesDRAM * t / RAM_BytesPerFlop;
                                                if (df > CPUChip.BusBandwidth / RAM_BytesPerSecondPerFlop)
                                                            df = CPUChip.BusBandwidth / RAM_BytesPerSecondPerFlop;
                                                if (df > MemChip.BusBandwidth * t / RAM_BytesPerSecondPerFlop)
                                                            df = MemChip.BusBandwidth * t / RAM_BytesPerSecondPerFlop;
                                                if (BestCost > DollarsPerNode / df) {
                                                            MemChip.Number = t;
                                                            BestCost = DollarsPerNode / df;
                                                }
                                    }
                        }
```

```
                        DollarsPerNode = CPUChip.CostPerChip + MemChip.Number * DRAMCost;

                        // if the specified number of units exceeds the capacity of the ASIC, drop out of the optimization
                        if (qx.Units * 30. > CustomChipSqMM / CustomChipMTransistorCellSize)
                                DollarsPerNode = 1e200;

                        Table = new nCUBEPicture();
                        Table->Set(out, 1, MemChip.Number, CustomChipTech.MaxChipSize(), DRAMTech.MaxChipSize(), "cpu", "dram", "pad");
                }

                // identify bottleneck and derate FLOPs appropriately
                char *why = "FLOPS";
                double DeratedFlops = CPUChip.FLOPS * CPUChip.Number;
                if (DeratedFlops > (CPUChip.IOBandwidth3d + RouterChip.IOBandwidth3d) / IO_BytesPerSecondPerFlop) {
                        DeratedFlops = (CPUChip.IOBandwidth3d + RouterChip.IOBandwidth3d) / IO_BytesPerSecondPerFlop;
                        why = "IO";
                }
                if (DeratedFlops > (CPUChip.BytesDRAM + MemChip.BytesDRAM * MemChip.Number) / RAM_BytesPerFlop) {
                        DeratedFlops = (CPUChip.BytesDRAM + MemChip.BytesDRAM * MemChip.Number) / RAM_BytesPerFlop;
                        why = "RAM";
                }
                if (DeratedFlops > CPUChip.BusBandwidth / RAM_BytesPerSecondPerFlop) {
                        DeratedFlops = CPUChip.BusBandwidth / RAM_BytesPerSecondPerFlop;
                        why = "CPU Bus";
                }
                if (MemChip.Number > 0 && DeratedFlops > MemChip.BusBandwidth * MemChip.Number / RAM_BytesPerSecondPerFlop) {
                        DeratedFlops = MemChip.BusBandwidth * MemChip.Number / RAM_BytesPerSecondPerFlop;
                        why = "Mem Bus";
                }
                UnitsPerPetaFlop = qx.PerformanceTarget/DeratedFlops;

                if (out != NULL) {

        fprintf(out, "<table border=\"0\" width=\"140\" valign=\"top\">\n");
        fprintf(out, "<tr valign=\"top\" align=\"center\"><td width=\"114\" valign=\"top\">\n");
        fprintf(out, "<!---------- #1 - secondary navigation---------->\n");
//          fprintf(out, "<font color=white>Nav text2<br></font>");
                        fprintf(out, "<table align=center><tr><td><table><tr><td align=center>\n");
                        fprintf(out, "%s"
//                                              "$%s<br>"
                                                "%.01f nodes<br>"
                                                "%s FLOPs<br>"
                                                "limited by %s%s\n", NOut1a,
//                                              GreekSuffix(DollarsPerNode * UnitsPerPetaFlop),
                                                UnitsPerPetaFlop, GreekSuffix(qx.PerformanceTarget), why, NOut2);
                        int skew = 3;
                        int thsize = 50;
                        if (1) {
                                        double Cost = DollarsPerNode * UnitsPerPetaFlop;

                                        double x = sqrt(Cost/7.75e6);
                                        double scale = thsize / (x + 1);

                                        fprintf(out, "<table align=center><tr><td valign=bottom>\n");
                                        fprintf(out, "<table><tr><td align=center><img src=mon.gif width=%d height=%d></td></tr><tr><td
align=center>%sSystem<br>Cost%s</td></tr></table>\n", (int)(scale * x), (int)(skew * scale * x), NOut1, NOut2);
                                        fprintf(out, "</td><td valign=bottom>");
                                        fprintf(out, "<table><tr><td align=center><img src=mon.gif width=%d height=%d></td></tr><tr><td
align=center>%sRed<br>Storm%s</td></tr></table>\n", (int)(scale), (int)(skew * scale), NOut1, NOut2);
                                        fprintf(out, "</td></tr></table>");
                        }
                        if (1) {
                                        double TotalW = 0.0;
                                        if (CPUChip.Number > 0)
                                                TotalW += CPUChip.Number * CPUChip.Power();
                                        if (MemChip.Number > 0)
                                                TotalW += MemChip.Number * MemChip.Power();
                                        if (RouterChip.Number > 0)
                                                TotalW += RouterChip.Number * RouterChip.Power();
                                        TotalW *= UnitsPerPetaFlop;

                                        double x = sqrt(TotalW/1.55e6);
                                        double scale = thsize / (x + 1);

                                        fprintf(out, "<table align=center><tr><td valign=bottom>\n");
                                        fprintf(out, "<table><tr><td align=center><img alt=%sW src=pwr.gif width=%d height=%d></td></tr><tr><td
align=center>%sTotal<br>Power%s</td></tr></table>\n", GreekSuffix(TotalW), (int)(scale * x), (int)(skew * scale * x), NOut1, NOut2);
                                        fprintf(out, "</td><td valign=bottom>");
                                        fprintf(out, "<table><tr><td align=center><img alt=%sW src=pwr2.gif width=%d height=%d></td></tr><tr><td
align=center>%sRed<br>Storm%s</td></tr></table>\n", "1.55M", (int)(scale), (int)(skew * scale), NOut1, NOut2);
                                        fprintf(out, "</td></tr></table>");
                        }

                        if (0) {

                                        double Uptime = 40.0 * 100 / UnitsPerPetaFlop;

                                        double x = sqrt(Uptime);
                                        double scale = thsize / (x + 1);
                                        scale *= 1.5;

                                        fprintf(out, "<table align=center><tr><td valign=bottom>\n");
                                        fprintf(out, "<table><tr><td align=center><img alt=\"%.11f H\" src=time.gif width=%d
height=%d></td></tr><tr><td align=center>%sTime To<br>Failure%s</td></tr></table>\n", Uptime, (int)(scale * x), (int)(scale * x * 404 / 462),
NOut1, NOut2);
                                        fprintf(out, "</td><td valign=bottom>");
                                        fprintf(out, "<table><tr><td align=center><img alt=\"%s\" src=time.gif width=%d height=%d></td></tr><tr><td
align=center>%sRed<br>Storm%s</td></tr></table>\n", "40 H", (int)(scale), (int)(scale * 404 / 462), NOut1, NOut2);
                                        fprintf(out, "</td></tr></table>");
                        }

                        fprintf(out, "</td></tr></table></table>\n");
//          fprintf(out, "<font color=white>Nav text2<br></font>");
        fprintf(out, "</td></tr>\n");
        fprintf(out, "\n");
        fprintf(out, "<tr valign=\"top\" align=\"center\"><td width=\"114\" valign=\"top\">\n");
        fprintf(out, "<!---------- #2 - secondary navigation---------->\n");
//                      <a href=\"http://www.sandia.gov/partnerships/\">\n");
//          fprintf(out, "<font color=white>Nav text3</font>");
        fprintf(out, "</td></tr></table>\n");
        fprintf(out, "\n");
        fprintf(out, "</td>\n");
        fprintf(out, "<td valign=\"top\" align=\"center\" width=\"100%%\">\n");
```

```
        fprintf(out, "\n");
        fprintf(out, "<table border=\"0\" width=\"445\" align=\"center\" valign=\"top\">\n");
        fprintf(out, "<tr width=\"445\"><td width=\"445\">\n");
        fprintf(out, "\n");
        fprintf(out, "<table border=\"0\" width=\"100%%\" align=\"center\" valign=\"top\" cellspacing=\"0\" cellpadding=\"0\">\n");
        fprintf(out, "<tr>\n");
        fprintf(out, "<td valign=\"top\">\n");
        fprintf(out, "<!--------- top banner graphic/text goes here --------->\n");
        fprintf(out, "<img src=\"petaflopsplanner.gif\" width=\"183\" height=\"26\" hspace=\"0\" vspace=\"4\" alt=\"Programs\"></td>\n");
        fprintf(out, "\n");
        fprintf(out, "<td valign=\"top\" align=\"right\">\n");
        fprintf(out, "<a href=\"http://www.sandia.gov/Main.html\"><img border=\"0\" src=\"logo.gif\" align=\"center\" width=\"151\"
hspace=\"0\" height=\"60\" vspace=\"0\" alt=\"Sandia National Laboratories\"></a></td></tr>\n");
        fprintf(out, "<tr><td colspan=\"2\">\n");
        fprintf(out, "<hr>\n");
        fprintf(out, "</td></tr>\n");
        fprintf(out, "</table>\n");
        fprintf(out, "\n");
        fprintf(out, "<center>\n");
        fprintf(out, "\n");
        fprintf(out, "<!---------- body begins here --------->\n");
        fprintf(out, "<table border=\"0\" width=\"100%%\" valign=\"top\" cellspacing=\"0\" cellpadding=\"0\">\n");
        fprintf(out, "<tr>\n");
        fprintf(out, "<td>\n");
//fprintf(out, "body");
                        fprintf(out, "<center>%sSilicon Area Floorplan%s</center>\n", Index1a, Index2);

                        Table->Print();

                        int nn =  (int)pow(UnitsPerPetaFlop, 1./3.);
                        if (0 && nn < 40) {
                                int n = nn/2;
                                if (n == 0) n = 1;

                                int w = 12;
                                int h = 6;

                                char *b = "x";
                                fprintf(out, "<table align=center><tr><th>%s%d x %d x %d System%s</th></tr>", Index1, nn, nn, nn, Index2);
                                fprintf(out, "<tr><td>");
                                fprintf(out, "<table border=0 cellpadding=0 cellspacing=0 align=center>");
                                if (1) for (int i = 0; i < n; i++) {
                                        fprintf(out, "<tr>");
                                        if (1) for (int j = 0; j < n - i - 1; j++)
                                                fprintf(out, "<td><img src=%sw.gif width=%d height=%d></td>", b, w, h);
                                        fprintf(out, "<td><img src=%sa.gif width=%d height=%d></td>", b, w, h);
                                        if (1) for (int j = 0; j < n - 1; j++)
                                                fprintf(out, "<td><img src=%sb.gif width=%d height=%d></td>", b, w, h);
                                        fprintf(out, "<td><img src=%sc.gif width=%d height=%d></td>", b, w, h);
                                        if (1) for (int j = 0; j < i; j++)
                                                fprintf(out, "<td><img src=%se.gif width=%d height=%d></td>", b, w, h);
                                        fprintf(out, "</tr>\n");
                                }
                                if (1) for (int i = 0; i < n; i++) {
                                        fprintf(out, "<tr>");
                                        if (1) for (int j = 0; j < n; j++)
                                                fprintf(out, "<td><img src=%sf.gif width=%d height=%d></td>", b, w, h);
                                        if (1) for (int j = 0; j < n - i - 1; j++)
                                                fprintf(out, "<td><img src=%se.gif width=%d height=%d></td>", b, w, h);
                                        fprintf(out, "<td><img src=%sg.gif width=%d height=%d></td>", b, w, h);
                                        fprintf(out, "</tr>\n");
                                }
                                fprintf(out, "</table>\n");
                                fprintf(out, "</td></tr></table>\n");

                        }

                        fprintf(out, "<table border align=center><tr><td valign=top>%s\n", Index1);

#define COST 0
#define SAGE 0
#define EXTPERF 0
#define POWER 1

#if COST                                                                                                        // {
                        fprintf(out, "<h2>Cost</h2>\n");
                        if (CPUChip.Number > 0) fprintf(out, "$%.2lf %s (%d x $%.2lf)<br>\n", CPUChip.Number * CPUChip.CostPerChip,
CPUChip.Name, CPUChip.Number, CPUChip.CostPerChip);
                        if (MemChip.Number > 0) fprintf(out, "$%.2lf %s (%d x $%.2lf)<br>\n", MemChip.Number * MemChip.CostPerChip,
MemChip.Name, MemChip.Number, MemChip.CostPerChip);
                        if (RouterChip.Number > 0) fprintf(out, "$%.2lf %s (%d x $%.2lf)<br>\n", RouterChip.Number * RouterChip.CostPerChip,
RouterChip.Name, RouterChip.Number, RouterChip.CostPerChip);
                        fprintf(out, "------------<br>$%.2lf chip per node total<br>\n", CPUChip.Number * CPUChip.CostPerChip + MemChip.Number
* MemChip.CostPerChip + RouterChip.Number * RouterChip.CostPerChip);
#endif                                                                                                                  // }

#if SAGE                                                                                                        // {
                        fprintf(out, "<h2>Sage Performance</h2>\n");

                        // parameters
                        double E = (MemChip.Number * MemChip.BytesDRAM + CPUChip.BytesDRAM)/100000;         // points per PE
                        double P = UnitsPerPetaFlop;        // number of PEs
                        double L = pow(E * P, .3333);       // points on side of cube
                        fprintf(out, "%s points per PE; %s PEs; %s points per side<br>\n", GreekSuffix(E), GreekSuffix(P), GreekSuffix(L));

                        double SurfaceZ = min(L*L, E/2);// points on Z surface
        double SurfaceY = 2*L;                          // points on Y surface
        double SurfaceX = 4;                            // points on X surface
                        fprintf(out, "%sx%sx%s slab<br>\n",GreekSuffix(SurfaceZ), GreekSuffix(SurfaceY), GreekSuffix(SurfaceX));

                        double PSMP = qx.Units;                         // processors in SMP
                        double CL = 1;                                                  // communications links per node
                        fprintf(out, "%s processors per SMP; %s links between PEs<br>\n",GreekSuffix(PSMP), GreekSuffix(CL));

                        // MPI Latency at 2 nanoseconds plus 5000 clock cycles
                        // adjust for distance @ 2 ns/foot plus 20 clock cycles per hop
                        double MPILatency = 2e-9 + 5000/CPUChip.ClkMHz/1e6;
                        double MPIBandwidth = RouterChip.IOBandwidth3d + CPUChip.IOBandwidth3d;

                        // Inter CPU latency at 2000 clock cycles
                        double LocalLatency = 2000/CPUChip.ClkMHz/1e6;

// latency for a message of size S
#define Lc1(S) ((S)<64 ? 4.8e-6 : (S)<=256 ? 4.9e-6 : (S)<=8192 ? 13.5e-6 : 23.2e-6)
```

```
#define Lc2(S) ((S)<64 ? 6.10e-6 : (S)<=512 ? 6.44e-6 : 13.8e-6)
#define Lc(S, P) ((P)<=4 ? Lc1(S) : Lc2(S))

// time per byte for a mesage of size S
#define Bc1inv(S) ((S)<64 ? 0 : (S)<=256 ? 13.9e-9 : (S)<=8192 ? 1.04e-9 : 1.37e-9)
#define Bc2inv(S) ((S)<64 ? 0 : (S)<=512 ? 12.2e-9 : 8.30e-9)
#define Bcinv(S, P) ((P)<=4 ? Bc1inv(S) : Bc2inv(S))

                        double MPIReal8 = 8;
                        double MPIInt = 4;

// Tcomp is the grid time
// someday this should distinguish between cache and main memory
#define Tcomp(E) ((.36/13500.*CPUChip.ClkMHz/500)*E)

// Memory contention -- what is this?
#define Tmem(P) ((P) <= 2 ? 1.8e-6 : 4.8e-6)

// formulas
// communications time of a message of size S
#define Tcomm(S, P) (Lc(S, P) + S * Bcinv(S, P))

// topology dependent: not logarithmic in a mesh
#define Tallreduce(P) (120*2*log(P)/log(2)*Tcomm(4, P))

#define Tmemcon(P, E) (E)*Tmem(P)

// this is something like the number of PEs on the surface of a subgrid
// these could all be contending for a grid link
#define PEsurface 12

#define C(P, E) min(max(L*L/CL/PEsurface, 1), PSMP/CL)

#define TGSComm(P, E) (C(P, E) * \
                        (160*Tcomm(SurfaceZ*MPIReal8, P) + \
              17*Tcomm(SurfaceZ*MPIInt, P) + \
             160*Tcomm(SurfaceY*MPIReal8, P) + \
              17*Tcomm(SurfaceY*MPIInt, P) + \
             160*Tcomm(SurfaceX*MPIReal8, P) + \
              17*Tcomm(SurfaceX*MPIInt, P)))

#define Tcycle(P, E) Tcomp(E) + Tmemcon(P, E) + TGSComm(P, E) + Tallreduce(P)
                        fprintf(out, "Tcomp(%s) = %ss<br>\n", GreekSuffix(E), GreekSuffix(Tcomp(E)));
                        fprintf(out, "Tmemcon(%s, %s) = %ss<br>\n", GreekSuffix(P), GreekSuffix(E), GreekSuffix(Tmemcon(P, E)));
                        fprintf(out, "TGSComm(%s, %s) = %ss<br>\n", GreekSuffix(P), GreekSuffix(E), GreekSuffix(TGSComm(P, E)));
                        fprintf(out, "Tallreduce(%s) = %ss<br>\n", GreekSuffix(P), GreekSuffix(Tallreduce(P)));
                        double aa=Tcycle(P, E);
                        fprintf(out, "Tcycle=%ss<br>\n", GreekSuffix(aa));
                        fprintf(out, "efficiency=%.01f%%<br>\n", Tcomp(E)/aa*100);
                                                                                                                    // }
#endif
#if EXTPERF                                                                                                     // {
                        fprintf(out, "<h2>Application Performance</h2>\n");

                        FILE *f = fopen("arch.bat", "w+");
                        if (f == NULL)
                                fprintf(out, "couldn't write arch.bat<br>\n");
                        else {
                                fprintf(f, "dir/b > arch.out\n");

                                fprintf(f, "rem <bytes>%.5le</bytes>\n", (MemChip.Number * MemChip.BytesDRAM + CPUChip.BytesDRAM));
                                fprintf(f, "rem <nodes>%.5le</nodes>\n", UnitsPerPetaFlop);
                                fprintf(f, "rem <psmp>%d</psmp>\n", qx.Units);
                                fprintf(f, "rem <mpilatency>%.5le</mpilatency>\n", 2e-9 + 5000/CPUChip.ClkMHz/1e6);
                                fprintf(f, "rem <mpibytetime>%.5le</mpibytetime>\n", RouterChip.IOBandwidth3d + CPUChip.IOBandwidth3d);

                                fclose(f);

                                UINT rc = WinExec("arch.bat", SW_SHOWNORMAL);
                                if (rc <= 31)
                                        fprintf(out, "error: return code %d from subjob<br>\n", rc);
                                else {
                                        f = fopen("arch.out", "r");
                                        if (f == NULL)
                                                fprintf(out, "couldn't read arch.out<br>\n");
                                        else {
                                                fprintf(out, "<pre>\n");
                                                char buf[500];
                                                int r;
                                                while ((r = fread(buf, 1, 500, f)) > 0) {
                                                        fwrite(buf, 1, r, out);
                                                }
                                                fprintf(out, "</pre>\n");
/*
                                                if (fseek(f, 0, SEEK_END) != 0)
                                                        fprintf(out, "couldn't seek arch.out<br>\n");
                                                else {
                                                        int len = ftell(f);
                                                        fprintf(out, "successfully read %d bytes<br>\n", len);
                                                }
*/
                                        }
                                        fclose(f);
                                }
                        }
                                                                                                                    // }
#endif

#if POWER                                                                                                       // {
                        fprintf(out, "<h2>Power</h2>\n");

                        double TotalW = 0.0;
                        if (CPUChip.Number > 0) {
                                fprintf(out, "%sW %s (%d x %sW)<br>\n", GreekSuffix(CPUChip.Number * CPUChip.Power()), CPUChip.Name,
CPUChip.Number, GreekSuffix(CPUChip.Power()));
                                TotalW += CPUChip.Number * CPUChip.Power();
                        }
                        if (MemChip.Number > 0) {
                                fprintf(out, "%sW %s (%d x %sW)<br>\n", GreekSuffix(MemChip.Number * MemChip.Power()), MemChip.Name,
MemChip.Number, GreekSuffix(MemChip.Power()));
                                TotalW += MemChip.Number * MemChip.Power();
                        }
                        if (RouterChip.Number > 0) {
                                fprintf(out, "%sW %s (%d x %sW)<br>\n", GreekSuffix(RouterChip.Number * RouterChip.Power()),
RouterChip.Name, RouterChip.Number, GreekSuffix(RouterChip.Power()));
                                TotalW += RouterChip.Number * RouterChip.Power();
```

```
                        )
                        fprintf(out, "------------<br>%.2lfW per node total<br>\n", TotalW);
                        fprintf(out, "%sW IC power for %d nodes<br>\n", GreekSuffix(TotalW * (int)UnitsPerPetaFlop), (int)UnitsPerPetaFlop);
                        fprintf(out, "%sW est. total (2x)<br>\n", GreekSuffix(TotalW * (int)UnitsPerPetaFlop * 2));
#endif                                                                                                          // )

                        fprintf(out, "%s</td><td valign=top>%s\n", Index2, Index1);

                        CPUChip.Print(out);

                        fprintf(out, "%s</td></tr><tr><td valign=top>%s\n", Index2, Index1);

                        MemChip.Print(out);

                        fprintf(out, "%s</td><td valign=top>%s\n", Index2, Index1);

                        RouterChip.Print(out);

                        fprintf(out, "%s</td></tr></table>\n", Index2);

//                      fprintf(out, "</td></tr>\n");       // basic page layout

//                      fprintf(out, "</table></td></tr></table>\n");   // basic page layout
                        fprintf(out, "<br clear=all>\n");
        fprintf(out, "</td></tr>\n");
        fprintf(out, "</table>\n");
        fprintf(out, "</center>\n");
        fprintf(out, "\n");
        fprintf(out, "<!---------- body ends here --------->\n");
        fprintf(out, "</td></tr></table>\n");
        fprintf(out, "\n");
        fprintf(out, "</td></tr></table>\n");
        fprintf(out, "\n");
        fprintf(out, "<hr width=\"100%%\">\n");
        fprintf(out, "\n");
        fprintf(out, "<table align=\"top\" border=\"0\" width=\"100%%\">\n");
        fprintf(out, "<tr align=\"center\">\n");
        fprintf(out, "<td valign=\"top\" width=\"140\">\n");
        fprintf(out, "<table border=\"0\" width=\"140\">\n");
        fprintf(out, "<tr><td align=\"center\" width=\"120\" valign=\"top\">\n");
        fprintf(out, "<font size=\"-1\" color=\"#FFFFFF\">\n");
        fprintf(out, "<br></font>\n");
        fprintf(out, "</td>\n");
        fprintf(out, "<td width=\"20\">\n");
        fprintf(out, "<br>\n");
        fprintf(out, "</td></tr></table>\n");
        fprintf(out, "\n");
        fprintf(out, "</td>\n");
        fprintf(out, "\n");
        fprintf(out, "<td valign=\"top\" align=\"center\" width=\"100%%\">\n");
        fprintf(out, "\n");
        fprintf(out, "<table border=\"0\" width=\"100%%\" align=\"center\" valign=\"top\">\n");
        fprintf(out, "<tr valign=top><td>\n");
        fprintf(out, "<font size=\"-1\"><center>\n");
        fprintf(out, "<a href=\"#TOP\">Back to top of page</a> || \n");
        fprintf(out, "<a href=\"http://www.sandia.gov/feedback.htm\">Questions and Comments</a> || \n");
        fprintf(out, "<a href=\"http://www.sandia.gov/dis.htm\">Acknowledgment and Disclaimer</a></center></font>\n");
        fprintf(out, "</td></tr></table>\n");
        fprintf(out, "\n");
        fprintf(out, "</td></tr></table>\n");
        fprintf(out, "\n");
        fprintf(out, "</html>\n");
        fprintf(out, "\n");
            }

            delete Table;

            return rval;
}

int Summary(FILE *index, Query &qx, int doit) {

    fprintf(index, "<html><head><title>Petaflops Planner</title></head>\n");
    fprintf(index, "<body background=\"bkgrnd.gif\" text=\"#000000\" link=\"#003366\" vlink=\"#cc0033\" alink=\"#000000\">\n");
    fprintf(index, "<a name=\"TOP\"></a>\n");
    fprintf(index, "<table border=\"0\"  width=\"100%%\">\n");
    fprintf(index, "<tr valign=\"top\"><td>\n");
    fprintf(index, "<table border=\"0\" width=\"140\" valign=\"top\">\n");
    fprintf(index, "<tr valign=\"top\"><td width=\"140\" valign=\"top\">\n");
    fprintf(index, "\n");
//          fprintf(index, "1 nav");
    fprintf(index, "</td></tr>\n");
    fprintf(index, "</table>\n");
    fprintf(index, "\n");
    fprintf(index, "<table border=\"0\" width=\"140\" valign=\"top\">\n");
    fprintf(index, "<tr valign=\"top\" align=\"center\"><td width=\"114\" valign=\"top\">\n");
    fprintf(index, "<!---------- #1 - secondary navigation------------>\n");
//          fprintf(index, "2 nav");
            fprintf(index, "<a href=/index.htm target=query>%sIndex%s</a><br>", NOut1, NOut2);
            fprintf(index, "<a href=/help.htm>%sHelp%s</a>", NOut1, NOut2);
    fprintf(index, "</td></tr>\n");
    fprintf(index, "\n");
    fprintf(index, "<tr valign=\"top\" align=\"center\"><td width=\"114\" valign=\"top\">\n");
    fprintf(index, "<!---------- #2 - secondary navigation------------>\n");
//          fprintf(index, "3 nav");
    fprintf(index, "</td></tr></table>\n");
    fprintf(index, "\n");
    fprintf(index, "</td>\n");
    fprintf(index, "<td valign=\"top\" align=\"center\" width=\"100%%\">\n");
    fprintf(index, "\n");
    fprintf(index, "<table border=\"0\" width=\"445\" align=\"center\" valign=\"top\">\n");
    fprintf(index, "<tr width=\"445\"><td width=\"445\">\n");
    fprintf(index, "\n");
    fprintf(index, "<table border=\"0\" width=\"100%%\" align=\"center\" valign=\"top\" cellspacing=\"0\" cellpadding=\"0\">\n");
    fprintf(index, "<tr>\n");
    fprintf(index, "<td valign=\"top\">\n");
    fprintf(index, "<!--------- top banner graphic/text goes here --------->\n");
    fprintf(index, "<img src=\"petaflopsplanner.gif\" width=\"183\" height=\"26\" hspace=\"0\" vspace=\"4\" alt=\"Programs\"></td>\n");
    fprintf(index, "\n");
    fprintf(index, "<td valign=\"top\" align=\"right\">\n");
    fprintf(index, "<a href=\"http://www.sandia.gov/Main.html\"><img border=\"0\" src=\"logo.gif\" align=\"center\" width=\"151\" hspace=\"0\"
height=\"60\" vspace=\"0\" alt=\"Sandia National Laboratories\"></a></td></tr>\n");
    fprintf(index, "<tr><td colspan=\"2\">\n");
```

```
    fprintf(index, "<hr>\n");
    fprintf(index, "</td></tr>\n");
    fprintf(index, "</table>\n");
    fprintf(index, "\n");
    fprintf(index, "<center>\n");
    fprintf(index, "\n");
    fprintf(index, "<!---------- body begins here --------->\n");
    fprintf(index, "<table border=\"0\" width=\"100%%\" valign=\"top\" cellspacing=\"0\" cellpadding=\"0\">\n");
    fprintf(index, "<tr>\n");
    fprintf(index, "<td>\n");
//          fprintf(index, "body");
if (doit != 0) {
            fprintf(index, "<table border=0><tr>\n");

            for (qx.Class = 0; qx.Class < 2; qx.Class++) {

                        fprintf(index, "<td>\n");
                        fprintf(index, "<table border><tr><th colspan=5>%s%s%s</th></tr>"

            "<tr><th>%sYear%s</th><th>%sRISC<br>PIM%s</th><th>%sVLIW<br>PIM%s</th><th>%sRed<br>Storm%s</th><th>%snCUBE<br>BG/L%s</th></tr>\n",
                                    Out1, qx.Class == 0 ? "Data Intensive" : "FLOPS Intensive", Out2, Out1, Out2, Out1, Out2, Out1,
Out2, Out1, Out2, Out1, Out2);
                        for (qx.Year = qx.Year1; qx.Year <= qx.Year2; qx.Year += 1.0) {

                                    fprintf(stderr, "%d ", (int)qx.Year);

                                    int BestCPUs[4];
                                    double BestCost[4];
                                    char *BestURL[4];

                                    if (1) for (int core = 0; core < 4; core++) {

                                                qx.Type = core == 0 ? RISC : core == 1 ? VLIW : core == 2 ? RED : NCUBE;

                                                BestCPUs[core] = 0;
                                                BestCost[core] = 1e100;
                                                BestURL[core] = NULL;

                                                // iterate through units -- PIM CPUs or DRAM chips
                                                for (qx.Units = qx.Units1; qx.Units <= qx.Units2; qx.Units++) {

                                                            double DollarsPerNode;
                                                            double UnitsPerPetaFlop;
                                                            int Error = FloorPlanner(NULL, qx, DollarsPerNode, UnitsPerPetaFlop);

                                                            if (Error == 0 && BestCost[core] > DollarsPerNode * UnitsPerPetaFlop) {
                                                                        BestCPUs[core] = qx.Units;
                                                                        BestCost[core] = DollarsPerNode * UnitsPerPetaFlop;
                                                                        if (BestURL[core] != NULL) free(BestURL[core]);
                                                                        BestURL[core] = strdup(qx.OutURL());
                                                            }
                                                }
                                    }

                                    if (1)
                                                fprintf(index, "<tr><td>%s%d%s</td><td align=center>%s<a href=%s target=meef>%d"
#define showcost 0
#if showcost != 0
                                                            " ($%s)"
#endif
                                                            "</a>%s</td>"
                                                            "<td align=center>%s<a href=%s target=meef>%d"
#if showcost != 0
                                                            " ($%s)"
#endif
                                                            "</a>%s</td>"
                                                            "<td align=center>%s<a href=%s target=meef>%d"
#if showcost != 0
                                                            " ($%s)"
#endif
                                                            "</a>%s</td>"
                                                            "<td align=center>%s<a href=%s target=meef>%d"
#if showcost != 0
                                                            " ($%s)"
#endif
                                                            "</a>%s</td></tr>\n", Out1, (int)qx.Year, Out2,
                                                            BestCost[0] < BestCost[1] && BestCost[0] < BestCost[2] && BestCost[0] < BestCost[3]
? Index1a : Out1, BestURL[0], BestCPUs[0],
#if showcost != 0
                                                            GreekSuffix(BestCost[0]),
#endif
                                                            Out2,
                                                            BestCost[1] <= BestCost[0] && BestCost[1] < BestCost[2] && BestCost[1] <
BestCost[3] ? Index1a : Out1, BestURL[1], BestCPUs[1],
#if showcost != 0
                                                            GreekSuffix(BestCost[1]),
#endif
                                                            Out2,
                                                            BestCost[2] <= BestCost[1] && BestCost[2] <= BestCost[0] && BestCost[2] <
BestCost[3] ? Index1a : Out1, BestURL[2], BestCPUs[2],
#if showcost != 0
                                                            GreekSuffix(BestCost[2]),
#endif
                                                            Out2,
                                                            BestCost[3] <= BestCost[2] && BestCost[3] <= BestCost[1] && BestCost[3] <=
BestCost[0] ? Index1a : Out1, BestURL[3], BestCPUs[3],
#if showcost != 0
                                                            GreekSuffix(BestCost[3]),
#endif
                                                            Out2);
                                    else
                                                fprintf(index, "<tr><td>%s%d%s</td><td align=center>%s<a href=%s target=meef>%d"
                                                            "<br>($%s)"
                                                            "</a>%s</td>"
                                                            "<td align=center>%s<a href=%s target=meef>%d"
                                                            "<br>($%s)"
                                                            "</a>%s</td>"
                                                            "<td align=center>%s<a href=%s target=meef>%d"
                                                            "<br>($%s)"
                                                            "</a>%s</td>"
                                                            "<td align=center>%s<a href=%s target=meef>%d"
                                                            "<br>($%s)"
                                                            "</a>%s</td></tr>\n", Out1, (int)qx.Year, Out2,
```

```
                                                        BestCost[0] < BestCost[1] && BestCost[0] < BestCost[2] && BestCost[0] < BestCost[3]
? Index1a : Out1, BestURL[0], BestCPUs[0],
                                                        GreekSuffix(BestCost[0]),
                                                        Out2,
                                                        BestCost[1] <= BestCost[0] && BestCost[1] < BestCost[2] && BestCost[1] <
BestCost[3] ? Index1a : Out1, BestURL[1], BestCPUs[1],
                                                        GreekSuffix(BestCost[1]),
                                                        Out2,
                                                        BestCost[2] <= BestCost[1] && BestCost[2] <= BestCost[0] && BestCost[2] <
BestCost[3] ? Index1a : Out1, BestURL[2], BestCPUs[2],
                                                        GreekSuffix(BestCost[2]),
                                                        Out2,
                                                        BestCost[3] <= BestCost[2] && BestCost[3] <= BestCost[1] && BestCost[3] <=
BestCost[0] ? Index1a : Out1, BestURL[3], BestCPUs[3],
                                                        GreekSuffix(BestCost[3]),
                                                        Out2);

                                    if (1) for (int core = 0; core < 4; core++)
                                                if (BestURL[core] != NULL)
                                                        free(BestURL[core]);
                                }
                                fprintf(stderr, "\n");

                                fprintf(index, "</table>\n");
                                fprintf(index, "</td>\n");
                        }

                        fprintf(index, "</tr>\n");
                        fprintf(index, "<tr><td colspan=2>%sLegend: n ($p) where<br>"
                                        "n is number of CPU units<br>"
                                        "$p is dollars to build a %sFLOPS supercomputer<br>%s"
                                        "%slarge text%s%s indicates this is the best alternative<br>%s</td></tr>",
                                        NOut1, GreekSuffix(qx.PerformanceTarget), NOut2, NOut1a, NOut2, NOut1, NOut2);

                        fprintf(index, "</table>\n");
}
                                qx.PrintQuery(index);
//fprintf(index, "body end");
                fprintf(index, "</td></tr>\n");
                fprintf(index, "</table>\n");
                fprintf(index, "</center>\n");
                fprintf(index, "\n");
                fprintf(index, "<!---------- body ends here --------->\n");
                fprintf(index, "</td></tr></table>\n");
                fprintf(index, "\n");
                fprintf(index, "</td></tr></table>\n");
                fprintf(index, "\n");
                fprintf(index, "<hr width=\"100%%\">\n");
                fprintf(index, "\n");
                fprintf(index, "<table align=\"top\" border=\"0\" width=\"100%%\">\n");
                fprintf(index, "<tr align=\"center\">\n");
                fprintf(index, "<td valign=\"top\" width=\"140\">\n");
                fprintf(index, "<table border=\"0\" width=\"140\">\n");
                fprintf(index, "<tr><td align=\"center\" width=\"120\" valign=\"top\">\n");
                fprintf(index, "<font size=\"-1\" color=\"#FFFFFF\">\n");
                fprintf(index, "<br></font>\n");
                fprintf(index, "</td>\n");
                fprintf(index, "<td width=\"20\">\n");
                fprintf(index, "<br>\n");
                fprintf(index, "</td></tr></table>\n");
                fprintf(index, "\n");
                fprintf(index, "</td>\n");
                fprintf(index, "\n");
                fprintf(index, "<td valign=\"top\" align=\"center\" width=\"100%%\">\n");
                fprintf(index, "\n");
                fprintf(index, "<table border=\"0\" width=\"100%%\" align=\"center\" valign=\"top\">\n");
                fprintf(index, "<tr valign=top><td>\n");
                fprintf(index, "<font size=\"-1\"><center>\n");
                fprintf(index, "<a href=\"#TOP\">Back to top of page</a> || \n");
                fprintf(index, "<a href=\"http://www.sandia.gov/feedback.htm\">Questions and Comments</a> || \n");
                fprintf(index, "<a href=\"http://www.sandia.gov/dis.htm\">Acknowledgment and Disclaimer</a></center></font>\n");
                fprintf(index, "</td></tr></table>\n");
                fprintf(index, "\n");
                fprintf(index, "</td></tr></table>\n");
                fprintf(index, "\n");
                fprintf(index, "</html>\n");
                fprintf(index, "\n");



                return 0;
}

int Help(FILE *index) {

    fprintf(index, "<html><head><title>Petaflops Planner Help</title></head>\n");
    fprintf(index, "<body background=\"bkgrnd.gif\" text=\"#000000\" link=\"#003366\" vlink=\"#cc0033\" alink=\"#000000\">\n");
    fprintf(index, "<a name=\"TOP\"></a>\n");
    fprintf(index, "<table border=\"0\". width=\"100%%\">\n");
    fprintf(index, "<tr valign=\"top\"><td>\n");
    fprintf(index, "<table border=\"0\" width=\"140\" valign=\"top\">\n");
    fprintf(index, "<tr valign=\"top\"><td width=\"140\" valign=\"top\">\n");
    fprintf(index, "\n");
//          fprintf(index, "1 nav");
    fprintf(index, "</td></tr>\n");
    fprintf(index, "</table>\n");
    fprintf(index, "\n");
    fprintf(index, "<table border=\"0\" width=\"140\" valign=\"top\">\n");
    fprintf(index, "<tr valign=\"top\" align=\"center\"><td width=\"114\" valign=\"top\">\n");
    fprintf(index, "<!---------- #1 - secondary navigation---------->\n");
//          fprintf(index, "2 nav");
            fprintf(index, "<a href=/index.htm target=query>%sIndex%s</a><br>", NOut1, NOut2);
            fprintf(index, "<a href=/help.htm>%sHelp%s</a>", NOut1, NOut2);
    fprintf(index, "</td></tr>\n");
    fprintf(index, "\n");
    fprintf(index, "<tr valign=\"top\" align=\"center\"><td width=\"114\" valign=\"top\">\n");
    fprintf(index, "<!---------- #2 - secondary navigation---------->\n");
//          fprintf(index, "3 nav");
    fprintf(index, "</td></tr></table>\n");
    fprintf(index, "\n");
    fprintf(index, "</td>\n");
    fprintf(index, "<td valign=\"top\" align=\"center\" width=\"100%%\">\n");
    fprintf(index, "\n");
    fprintf(index, "<table border=\"0\" width=\"445\" align=\"center\" valign=\"top\">\n");
```

```
    fprintf(index, "<tr width=\"445\"><td width=\"445\">\n");
    fprintf(index, "\n");
    fprintf(index, "<table border=\"0\" width=\"100%%\" align=\"center\" valign=\"top\" cellspacing=\"0\" cellpadding=\"0\">\n");
    fprintf(index, "<tr>\n");
    fprintf(index, "<td valign=\"top\">\n");
    fprintf(index, "<!--------- top banner graphic/text goes here --------->\n");
    fprintf(index, "<img src=\"petaflopsplanner.gif\" width=\"183\" height=\"26\" hspace=\"0\" vspace=\"4\" alt=\"Programs\"></td>\n");
    fprintf(index, "\n");
    fprintf(index, "<td valign=\"top\" align=\"right\">\n");
    fprintf(index, "<a href=\"http://www.sandia.gov/Main.html\"><img border=\"0\" src=\"logo.gif\" align=\"center\" width=\"151\" hspace=\"0\"
height=\"60\" vspace=\"0\" alt=\"Sandia National Laboratories\"></a></td></tr>\n");
    fprintf(index, "<tr><td colspan=\"2\">\n");
    fprintf(index, "<hr>\n");
    fprintf(index, "</td></tr>\n");
    fprintf(index, "</table>\n");
    fprintf(index, "\n");
    fprintf(index, "<center>\n");
    fprintf(index, "\n");
    fprintf(index, "<!---------- body begins here --------->\n");
    fprintf(index, "<table border=\"0\" width=\"100%%\" valign=\"top\" cellspacing=\"0\" cellpadding=\"0\">\n");
    fprintf(index, "<tr>\n");
    fprintf(index, "<td>\n");
    fprintf(index, "%s\n", Out1a);
//          fprintf(index, "<h2>Help</h2>\n");

    fprintf(index, "Plans a parallel supercomputer with a specified performance level.<p>\n");

            fprintf(index, "<h3>Technology</h3>\n");

            fprintf(index, "The software plans a supercomputer based on projections of CMOS technology up to 15 years into the future.\n");
            fprintf(index, "Projections are based on the <a href=http://public.itrs.net>Semiconductor Industries Association's (SIA's)
International Technology Roadmap for Semiconductors (ITRS)</a>, which generalizes Moore's Law to dozens of parameters projected 15 years into
the future.<p>\n");

            fprintf(index, "<h3>Architectures</h3>\n");

            fprintf(index, "The software implements three well-known architectures:\n");
    fprintf(index, "<ul>\n");
    fprintf(index, "<li><img src=pimchip.gif align=right>Processor-In-Memory architecture whereby an entire node is implemented on a single
chip. "
                                "Each chip contains one or more microprocessor cores and DRAM. "
                                "The cores may be conventional CISC/RISC design or multiple issues such as VLIW or Superscalar.\n");
    fprintf(index, "<li><img src=redchip.gif align=right>Discrete components, as in ASCI Red. "
                                "Each node in this architecture is an Symmetric MultiProcessor (SMP) with one or more microprocessor chips,
one of more DRAM chips, and a ASIC wormhole router chip. "
                                "filler filler filler filler filler filler\n");
    fprintf(index, "<li><img src=ncube.gif align=right>Combined processor and router architecture, as in the nCUBE. "
                                "Each node contains one ASIC with a wormhole router and one or more CPU cores and DRAM chips.\n");
    fprintf(index, "</ul>\n");

    fprintf(index, "The software assumes a 3-d mesh interconnect.<p>\n");

            fprintf(index, "<h3>Design Optimization</h3>\n");

    fprintf(index, "The software attempts to balance the computers by five parameters:<p>\n");

    fprintf(index, "<ul>\n");
    fprintf(index, "<li>I/O bandwidth from each processor to its neighbor\n");
    fprintf(index, "<li>Cumulative I/O bandwidth from each processor to all neighbors combined\n");
    fprintf(index, "<li>Memory size\n");
    fprintf(index, "<li>Main memory bus bandwidth\n");
    fprintf(index, "<li>Technology derating factor\n");
    fprintf(index, "</ul>\n");

            fprintf(index, "<h4>Derated Flops</h4>\n");

    fprintf(index, "The software uses the concept of \"derated flops\" to find the most cost effective design.\n");
    fprintf(index, "Specifically, the software calculates the highest FLOP rate that could be claimed for each node while meeting all the
balance requirements.\n");
    fprintf(index, "The summary screen then reports the layout within each architecture providing the highest derated flops per cost.<p>\n");

    fprintf(index, "Each layout page includes a navigation section to permit the user to view designs using technology of different years and
with different numbers of processors per node.\n");
    fprintf(index, "Each of these design will be balanced, but will generally be of higher cost than the designed optimal design.<p>\n");

            fprintf(index, "<h4>Technology Derating Factor</h4>\n");

    fprintf(index, "The software uses a \"technology derating factor,\" which if a form of \"fudge factor.\"\n");
    fprintf(index, "It has been known for years that the best technology goes to commercial and consumer products.\n");
    fprintf(index, "Supercomputers tend to get technology a few years later, with the delay being related to the negotiating power of the
Government, the public relations value of science, etc. To account for this, the software derates the performance of microprocessors and ASICs
by a selectable factor (default .5).\n");
    fprintf(index, "Memories are not affected.<p>\n");

            fprintf(index, "<h4>Memory Size</h4>\n");

    fprintf(index, "The memory size parameter is given in bytes/FLOPs at a 1 teraflops cumulative processor speed.\n");
    fprintf(index, "In accordance with ASCI design rules, the parameter is multiplied by (target flops/1 teraflops)<sup>-.25</sup>.\n");
    fprintf(index, "(This factor is somewhat heuristic. However, the justification is based on scaling a 3-d finite difference equation by 2 in
linear dimension.\n");
    fprintf(index, "Such a scaling would require 2<sup>3</sup> more memory but 2<sup>4</sup> more computation because the timestep must be
divided along with linear dimension.)<p>\n");

            fprintf(index, "<h4>Power Consumption</h4>\n");

    fprintf(index, "The software reports, but does not optimize power consumption per chip and over the whole machine.\n");
    fprintf(index, "Power consumption per chip is based on equivalent real chips in 2002 scaled by CMOS power scaling rules.\n");

            fprintf(index, "<h3>Output Legend</h3>\n");

    fprintf(index, "<ul>\n");
    fprintf(index, "<li>Squares represent integrated circuits. The size of a square represents the size of the integrated circuit die, with the
scaling factor consistent across all screens.\n");
    fprintf(index, "<li>Integrated circuit I/O pins are represented by horizontal bar graphs. The overall scale of the graphs is consistent
across all screens.\n");
    fprintf(index, "<li>Yellow represents memory. Yellow squares represent DRAM chips. Yellow regions in bar graphs represent pins devoted to
the memory bus.\n");
            fprintf(index, "<li>Alternating red and blue represent processors. Red and blue squares represent processor chips or cores. Red and
blue regions in the bar graphs represent pins devoted to chip-to-chip interconnections.\n");
            fprintf(index, "<li>Black squares represent wormhome routers.\n");
    fprintf(index, "</ul>\n");

    fprintf(index, "%s\n", Out2);
        fprintf(index, "</td></tr>\n");
```

```
            fprintf(index, "</table>\n");
            fprintf(index, "</center>\n");
            fprintf(index, "\n");
            fprintf(index, "<!---------- body ends here --------->\n");
            fprintf(index, "</td></tr></table>\n");
            fprintf(index, "\n");
            fprintf(index, "</td></tr></table>\n");
            fprintf(index, "\n");
            fprintf(index, "<hr width=\"100%%\">\n");
            fprintf(index, "\n");
            fprintf(index, "<table align=\"top\" border=\"0\" width=\"100%%\">\n");
            fprintf(index, "<tr align=\"center\">\n");
            fprintf(index, "<td valign=\"top\" width=\"140\">\n");
            fprintf(index, "<table border=\"0\" width=\"140\">\n");
            fprintf(index, "<tr><td align=\"center\" width=\"120\" valign=\"top\">\n");
            fprintf(index, "<font size=\"-1\" color=\"#FFFFFF\">\n");
            fprintf(index, "<br></font>\n");
            fprintf(index, "</td>\n");
            fprintf(index, "<td width=\"20\">\n");
            fprintf(index, "<br>\n");
            fprintf(index, "</td></tr></table>\n");
            fprintf(index, "\n");
            fprintf(index, "</td>\n");
            fprintf(index, "\n");
            fprintf(index, "<td valign=\"top\" align=\"center\" width=\"100%%\">\n");
            fprintf(index, "\n");
            fprintf(index, "<table border=\"0\" width=\"100%%\" align=\"center\" valign=\"top\">\n");
            fprintf(index, "<tr valign=top><td>\n");
            fprintf(index, "<font size=\"-1\"><center>\n");
            fprintf(index, "<a href=\"#TOP\">Back to top of page</a> || \n");
            fprintf(index, "<a href=\"http://www.sandia.gov/feedback.htm\">Questions and Comments</a> || \n");
            fprintf(index, "<a href=\"http://www.sandia.gov/dis.htm\">Acknowledgment and Disclaimer</a></center></font>\n");
            fprintf(index, "</td></tr></table>\n");
            fprintf(index, "\n");
            fprintf(index, "</td></tr></table>\n");
            fprintf(index, "\n");
            fprintf(index, "</html>\n");
            fprintf(index, "\n");

            return 0;
}

void WebServe(SOCKET theClient, char *Path, Query &qx) {

            char *File = Path;
            if (1) for (char *x = Path; *x != 0; x++)
                        if (*x == '/')
                                    File = x + 1;

            // print the plain form
            if (stricmp(Path, "/") == 0 || stricmp(Path, "/index.htm") == 0) {
                        send(theClient, "HTTP/1.0 200\r\n", 14, 0);
                        send(theClient, "Content-type: ", 14, 0);
                        send(theClient, "text/html", 9, 0);
                        send(theClient, "\r\n", 2, 0);
                        send(theClient, "\r\n", 2, 0);

                        FILE *index = fopen("tmp.htm", "wb");
                        Summary(index, qx, 0);
                        fclose(index);
                        FILE *t = fopen("tmp.htm", "rb");
                        char buf[1234];
                        int i;
                        do {
                                    i = fread(buf, 1, sizeof buf, t);
                                    if (i > 0) send(theClient, buf, i, 0);
                        } while (i == sizeof buf);

                        fclose(t);
            }

            // print an output table with a form at the bottom
            else if (stricmp(Path, "/planner") == 0 && qx.Command == 2) {
                        send(theClient, "HTTP/1.0 200\r\n", 14, 0);
                        send(theClient, "Content-type: ", 14, 0);
                        send(theClient, "text/html", 9, 0);
                        send(theClient, "\r\n", 2, 0);
                        send(theClient, "\r\n", 2, 0);

                        FILE *index = fopen("tmp.htm", "wb");
                        Summary(index, qx, 1);
                        fclose(index);
                        FILE *t = fopen("tmp.htm", "rb");
                        char buf[1234];
                        int i;
                        do {
                                    i = fread(buf, 1, sizeof buf, t);
                                    if (i > 0) send(theClient, buf, i, 0);
                        } while (i == sizeof buf);

                        fclose(t);

            }

            // print help
            else if (stricmp(Path, "/help.htm") == 0) {
                        send(theClient, "HTTP/1.0 200\r\n", 14, 0);
                        send(theClient, "Content-type: ", 14, 0);
                        send(theClient, "text/html", 9, 0);
                        send(theClient, "\r\n", 2, 0);
                        send(theClient, "\r\n", 2, 0);

                        FILE *index = fopen("tmp.htm", "wb");
                        Help(index);
                        fclose(index);
                        FILE *t = fopen("tmp.htm", "rb");
                        char buf[1234];
                        int i;
                        do {
                                    i = fread(buf, 1, sizeof buf, t);
                                    if (i > 0) send(theClient, buf, i, 0);
                        } while (i == sizeof buf);

                        fclose(t);
```

```
		}

		// access a single floorplan
		else if (stricmp(Path, "/planner") == 0 && qx.Command == 1) {
			send(theClient, "HTTP/1.0 200\r\n", 14, 0);
			send(theClient, "Content-type: ", 14, 0);
			send(theClient, "text/html", 9, 0);
			send(theClient, "\r\n", 2, 0);
			send(theClient, "\r\n", 2, 0);

			double DollarsPerNode = 0.0;
			double UnitsPerPetaFlop = 0.0;

			FILE *index = fopen("tmp.htm", "wb");
			FloorPlanner(index, qx, DollarsPerNode, UnitsPerPetaFlop);
			fclose(index);
			FILE *t = fopen("tmp.htm", "rb");
			char buf[1234];
			int i;
			do {
				i = fread(buf, 1, sizeof buf, t);
				if (i > 0) send(theClient, buf, i, 0);
			} while (i == sizeof buf);
			fclose(t);
		}
	else if (stricmp(File, "sandia.gif") == 0) {
		static unsigned char sandia[1780] = { 71, 73, 70, 56, 57, 97, 151, 0, 60, 0, 179, 255, 0, 192, 192, 192, 204, 255, 255, 204, 204, 255,
204, 204, 204, 153, 204, 255, 153, 204, 204, 153, 153, 153, 102, 204, 204, 102, 153, 204, 102, 102, 102, 51, 153, 204, 51, 51, 51, 0, 153, 204,
0, 0, 0, 0, 0, 0, 0, 0, 0, 33, 249, 4, 1, 0, 0, 0, 0, 44, 0, 0, 0, 0, 151, 0, 60, 0, 0, 4, 255, 16, 200, 73, 171, 189, 56, 235, 205, 187, 255,
96, 248, 5, 129, 96, 18, 104, 170, 174, 132, 41, 144, 129, 40, 207, 116, 13, 18, 197, 129, 32, 74, 239, 255, 192, 160, 240, 183, 67, 28, 10,
49, 217, 96, 105, 107, 58, 55, 1, 222, 112, 74, 173, 246, 24, 10, 66, 104, 144, 104, 120, 19, 3, 219, 192, 96, 0, 112, 203, 79, 27, 193, 202,
110, 11, 17, 73, 206, 192, 219, 88, 208, 195, 52, 131, 23, 160, 95, 164, 107, 5, 110, 130, 131, 10, 113, 26, 122, 13, 97, 115, 13, 104, 51,
139, 102, 11, 9, 127, 51, 107, 132, 150, 86, 8, 29, 93, 126, 18, 118, 146, 52, 143, 147, 52, 1, 152, 70, 71, 5, 168, 44, 44, 168, 168, 7, 58,
82, 84, 5, 28, 155, 19, 6, 9, 104, 182, 9, 96, 18, 100, 185, 187, 188, 186, 136, 102, 183, 192, 186, 120, 162, 24, 129, 66, 5, 47, 162, 36, 4,
7, 67, 179, 94, 199,
				181, 116, 117, 157, 215, 117, 97, 93, 218, 124, 216, 221, 119, 200, 24, 66, 7, 227, 22, 149, 64, 90, 26, 139, 11, 6, 213, 158,
3, 158, 0, 118, 11, 99, 94, 239, 248, 92, 123, 125, 139, 239, 180, 231, 40, 164, 243, 177, 46, 3, 9, 23, 170, 76, 24, 218, 0, 171, 71, 166, 13,
136, 240, 73, 144, 103, 239, 155, 159, 121, 0, 186, 220, 194, 246, 173, 129, 69, 138, 97, 255, 16, 85, 59, 23, 13, 8, 20, 54, 12, 100, 121, 24,
216, 163, 131, 60, 58, 159, 40, 244, 161, 215, 224, 147, 198, 153, 29, 45, 86, 16, 25, 112, 66, 67, 5, 230, 52, 144, 98, 163, 178, 195, 208,
31, 2, 60, 236, 99, 52, 204, 142, 151, 139, 53, 37, 220, 228, 40, 108, 38, 23, 167, 212, 122, 74, 8, 82, 20, 131, 128, 54, 93, 57, 4, 73, 250,
161, 94, 198, 47, 180, 48, 78, 229, 84, 21, 155, 83, 93, 89, 181, 6, 41, 232, 21, 44, 136, 185, 25, 94, 54, 66, 196, 115, 166, 218, 154, 56,
219, 218, 203, 202, 179, 231, 81, 31, 100, 13, 146, 0, 240, 117, 74, 80, 15, 63, 195, 82, 88, 20, 179, 79, 96, 183, 81, 207, 110, 100, 203,
175, 142, 176, 142, 35, 145, 29, 238, 177, 80, 67, 227, 33, 143, 59, 148, 244, 33, 153, 130, 217, 108, 238, 56, 154, 253, 187, 49, 81, 78, 203,
89, 187, 133, 118, 22,
				164, 116, 134, 211, 111, 64, 172, 238, 145, 218, 2, 162, 72, 78, 203, 212, 115, 10, 213, 102, 212, 122, 9, 156, 126, 124, 138,
117, 247, 164, 209, 10, 64, 0, 15, 242, 208, 131, 50, 31, 221, 49, 24, 192, 202, 180, 163, 187, 5, 126, 186, 160, 209, 8, 9, 31, 122, 157, 199,
199, 87, 52, 28, 68, 160, 42, 21, 195, 133, 220, 87, 17, 231, 187, 67, 151, 239, 84, 192, 196, 255, 7, 214, 77, 84, 224, 56, 163, 133, 119,
137, 27, 12, 20, 196, 82, 120, 90, 69, 8, 66, 130, 20, 44, 232, 134, 131, 64, 64, 40, 225, 134, 39, 17, 81, 161, 133, 68, 77, 176, 93, 118, 24,
112, 97, 204, 68, 39, 106, 242, 75, 19, 41, 58, 49, 218, 99, 216, 129, 184, 140, 136, 245, 149, 8, 211, 68, 113, 109, 224, 143, 19, 59, 62,
241, 226, 4, 49, 202, 8, 68, 81, 35, 230, 69, 7, 39, 139, 20, 120, 6, 36, 243, 137, 33, 145, 143, 64, 192, 40, 100, 21, 68, 214, 120, 193, 34,
18, 37, 153, 129, 29, 141, 164, 209, 163, 139, 81, 2, 57, 101, 44, 52, 154, 100, 99, 29, 81, 105, 105, 6, 25, 120, 232, 165, 72, 53, 75, 28,
195, 4, 155, 147, 145, 49, 193, 151, 77, 252, 40, 65, 144, 99, 2, 37, 166, 153, 87, 226, 179, 135, 150, 228, 129, 161, 77, 95, 215, 124, 242,
212, 141, 47, 49, 250, 36,
				152, 63, 60, 54, 34, 105, 48, 252, 52, 23, 12, 254, 249, 32, 37, 160, 22, 36, 169, 15, 53, 34, 185, 233, 73, 62, 137, 248, 163,
23, 0, 79, 129, 182, 201, 61, 165, 62, 154, 103, 152, 18, 76, 26, 158, 165, 64, 36, 198, 18, 113, 127, 254, 96, 100, 34, 241, 128, 202, 17,
123, 92, 154, 225, 43, 103, 80, 161, 129, 143, 122, 194, 50, 229, 170, 13, 122, 50, 22, 156, 255, 79, 85, 216, 90, 78, 174, 62, 236, 186, 143,
161, 182, 81, 0, 44, 83, 73, 178, 167, 83, 176, 168, 150, 135, 163, 178, 226, 190, 26, 41, 181, 224, 81, 64, 43, 82, 19, 220, 234, 231, 158,
86, 118, 58, 172, 150, 87, 221, 24, 108, 183, 153, 249, 85, 94, 150, 209, 209, 81, 198, 178, 53, 52, 27, 227, 172, 85, 20, 52, 233, 166, 186,
158, 185, 132, 160, 173, 122, 54, 207, 189, 212, 120, 171, 175, 177, 140, 32, 2, 23, 185, 93, 50, 11, 43, 0, 3, 127, 72, 133, 193, 51, 194,
203, 169, 128, 89, 121, 50, 44, 47, 207, 113, 27, 113, 190, 152, 81, 60, 94, 121, 193, 2, 60, 202, 198, 216, 49, 160, 96, 193, 34, 98, 49, 100,
153, 9, 7, 106, 91, 56, 137, 120, 187, 109, 25, 73, 234, 241, 201, 196, 18, 184, 167, 50, 198, 105, 8, 28, 101, 1, 116, 89, 81, 80, 0, 168,
236, 28, 107, 188, 36, 219, 134, 101,
				34, 70, 103, 35, 9, 196, 92, 83, 149, 178, 203, 96, 255, 91, 174, 198, 231, 138, 76, 144, 5, 82, 179, 253, 67, 149, 35, 79, 22,
87, 117, 22, 51, 119, 150, 59, 161, 126, 193, 23, 77, 46, 111, 210, 47, 211, 80, 166, 189, 213, 15, 116, 13, 78, 102, 5, 86, 59, 27, 247, 157,
232, 225, 33, 95, 69, 221, 232, 18, 73, 83, 140, 52, 222, 158, 103, 217, 160, 193, 229, 255, 82, 209, 185, 67, 79, 147, 230, 166, 59, 129, 58,
21, 240, 201, 154, 219, 107, 95, 189, 56, 129, 7, 118, 218, 250, 157, 175, 163, 45, 186, 225, 61, 132, 101, 122, 237, 168, 247, 128, 225, 234,
28, 246, 78, 1, 133, 208, 158, 254, 187, 21, 182, 215, 218, 110, 134, 190, 39, 143, 1, 118, 234, 190, 93, 58, 241, 207, 179, 43, 65, 166, 26,
78, 246, 122, 156, 202, 195, 30, 2, 243, 62, 233, 252, 46, 186, 211, 82, 0, 28, 3, 137, 101, 90, 156, 220, 103, 91, 224, 173, 242, 56, 141,
208, 219, 4, 254, 21, 119, 251, 247, 170, 43, 176, 67, 28, 249, 157, 127, 167, 204, 218, 102, 150, 134, 46, 31, 104, 159, 7, 176, 227, 155,
197, 12, 175, 10, 231, 243, 13, 0, 34, 179, 171, 140, 169, 207, 127, 78, 232, 26, 235, 182, 55, 22, 15, 76, 106, 99, 28, 96, 224, 153, 50, 54,
134, 70, 176, 103, 64, 119, 10, 144, 129, 222,
				180, 38, 17, 238, 163, 73, 99, 144, 19, 19, 72, 56, 17, 17, 214, 194, 133, 0, 192, 75, 7, 46, 40, 56, 177, 144, 110, 131, 140,
59, 18, 55, 234, 240, 150, 9, 132, 3, 114, 53, 233, 2, 171, 62, 117, 141, 48, 96, 165, 34, 70, 139, 142, 85, 176, 178, 139, 70, 109, 67, 2, 26,
228, 0, 13, 53, 53, 192, 10, 226, 208, 107, 39, 60, 11, 96, 130, 101, 25, 145, 255, 68, 164, 54, 100, 224, 146, 45, 48, 119, 19, 90, 68, 132,
12, 152, 185, 197, 170, 196, 136, 145, 159, 84, 239, 2, 2, 240, 158, 16, 108, 102, 193, 247, 93, 113, 71, 69, 19, 91, 122, 202, 163, 30, 158,
0, 132, 61, 93, 163, 151, 175, 152, 210, 143, 172, 248, 99, 15, 119, 155, 158, 29, 161, 224, 138, 34, 56, 178, 145, 173, 185, 64, 166, 72, 116,
69, 51, 180, 201, 87, 71, 107, 153, 84, 0, 195, 17, 236, 105, 230, 27, 146, 80, 147, 29, 238, 65, 44, 65, 50, 34, 75, 34, 28, 209, 27, 199,
113, 43, 253, 237, 175, 92, 78, 12, 27, 177, 248, 182, 201, 205, 104, 139, 137, 160, 76, 22, 30, 70, 121, 153, 216, 4, 81, 55, 17, 25, 201, 27,
20, 56, 137, 73, 102, 129, 29, 143, 138, 136, 83, 72, 137, 178, 98, 213, 18, 39, 141, 234, 97, 32, 231, 86, 49, 61, 70, 164, 136, 29, 201, 150,
4, 220, 101, 63,
				87, 164, 162, 5, 47, 128, 1, 20, 96, 112, 2, 28, 228, 96, 93, 171, 28, 23, 26, 240, 120, 178, 111, 241, 145, 147, 156, 88, 213,
39, 167, 153, 45, 94, 234, 209, 84, 158, 28, 198, 163, 230, 215, 167, 42, 164, 68, 71, 79, 50, 229, 61, 50, 233, 76, 205, 224, 68, 104, 81,
161, 231, 46, 101, 217, 76, 53, 177, 201, 78, 52, 225, 196, 54, 251, 185, 32, 87, 162, 111, 161, 157, 132, 105, 103, 215, 144, 117, 22, 52,
198, 51, 51, 243, 144, 160, 33, 53, 186, 141, 140, 214, 228, 87, 28, 1, 31, 69, 253, 25, 201, 172, 193, 4, 58, 119, 232, 134, 47, 203, 192, 23,
96, 162, 180, 38, 210, 252, 194, 9, 95, 230, 135, 94, 210, 228, 22, 245, 160, 204, 203, 98, 194, 179, 149, 90, 193, 162, 46, 197, 135, 50, 87,
134, 158, 84, 101, 195, 95, 58, 185, 28, 154, 62, 177, 76, 154, 64, 213, 167, 78, 68, 226, 53, 64, 39, 144, 117, 25, 85, 1, 72, 0, 65, 156,
176, 151, 66, 75, 154, 245, 172, 33, 52, 33, 8, 215, 52, 86, 237, 181, 16, 132, 112, 149, 19, 12, 151, 0, 195, 229, 9, 0, 7, 175, 152, 210, 14,
78, 65, 0, 98, 102, 239, 175, 51, 128, 65, 9, 92, 64, 216, 194, 134, 211, 128, 128, 77, 108, 7, 34, 0, 0, 59, };
		BigGIFHelper(theClient, sandia, 1780);
	}
	else if (stricmp(File, "petaflopsplanner.gif") == 0) {
		static unsigned char petaflopsplanner[1418] = { 71, 73, 70, 56, 55, 97, 183, 0, 26, 0, 247, 0, 0, 0, 0, 0, 128, 0, 0, 0, 128, 0, 128,
128, 0, 0, 0, 128, 128, 0, 128, 0, 128, 128, 192, 192, 192, 128, 128, 128, 255, 0, 0, 0, 255, 0, 255, 255, 0, 0, 0, 255, 255, 0, 255, 0, 255,
255, 255, 255, 255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
				0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
				0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
				0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 44, 0, 0, 0, 0, 183, 0, 26, 0, 0, 8, 254, 0, 9, 8, 28, 72, 80, 224, 131, 131, 8, 19, 42,
68, 72, 96, 161, 67, 134, 5, 13, 62, 156, 72, 177, 162, 195, 136, 17, 33, 74, 180, 200, 177, 163, 199, 139, 24, 11, 122, 28, 120, 160, 99, 72,
2, 37, 63, 170, 172, 120, 146, 224, 65, 130, 41, 87, 202, 156, 169, 177, 101, 67, 139, 48, 57, 182, 140, 73, 147, 166, 205, 129, 15, 114, 246,
```

```
28, 250, 241, 39, 80, 150, 36, 117, 146, 60, 32, 148, 168, 76, 163, 17, 121, 58, 157, 10, 82, 224, 129, 171, 88, 11, 74, 93, 216, 148, 98, 206,
174, 84, 77, 46, 197, 202, 52, 228, 214, 176, 97, 193, 6, 77, 186, 246, 232, 201, 182, 46, 127, 242, 20, 153, 208, 37, 220, 155, 53, 143, 114,
101, 91, 23, 99, 74, 140, 53, 95, 218, 133, 43, 88, 111, 222, 190, 6, 227, 242, 173, 89, 22, 170, 99, 179, 133, 51, 70, 14, 57, 89, 50, 98,
148, 123, 163, 222, 181, 11, 19, 48, 220, 198, 131, 223, 110, 38, 9, 246, 177, 105, 185, 163, 57, 159, 246, 204, 88, 225, 73, 208, 128, 141,
166, 22, 105, 115, 54, 108, 168, 89, 73, 235, 182, 122, 85, 43, 76, 208, 176, 115, 3, 213, 42, 156, 120, 110, 172, 151, 87, 23, 71, 105, 220,
        247, 239, 224, 207, 59, 63, 239, 109, 250, 170, 70, 224, 75, 25, 127, 221, 141, 50, 166, 116, 190, 144, 19, 254, 90, 119, 109,
244, 118, 119, 237, 217, 35, 3, 247, 238, 151, 61, 243, 174, 229, 225, 251, 125, 128, 28, 189, 213, 212, 227, 243, 166, 175, 108, 248, 97, 249,
245, 249, 125, 38, 159, 85, 3, 106, 149, 220, 126, 179, 241, 102, 222, 78, 153, 113, 87, 82, 120, 7, 222, 23, 33, 93, 85, 153, 101, 93, 105,
206, 73, 168, 94, 129, 213, 145, 167, 32, 89, 127, 61, 22, 162, 89, 219, 105, 230, 161, 129, 151, 45, 136, 87, 138, 32, 230, 87, 162, 95, 5, 2,
55, 97, 121, 39, 158, 197, 31, 131, 196, 125, 135, 25, 132, 250, 105, 168, 30, 125, 148, 213, 232, 21, 119, 221, 233, 56, 151, 131, 51, 162,
150, 226, 68, 211, 129, 104, 228, 133, 72, 242, 120, 163, 84, 77, 81, 55, 216, 143, 76, 182, 247, 32, 146, 246, 97, 214, 99, 119, 45, 202, 184,
164, 127, 139, 165, 216, 148, 145, 2, 246, 167, 227, 138, 40, 138, 23, 227, 144, 38, 166, 233, 35, 154, 95, 218, 136, 161, 143, 13, 246, 181,
33, 91, 25, 238, 72, 100, 94, 183, 49, 86, 150, 144, 173, 101, 105, 98, 159, 71, 18, 184, 152, 142, 122, 126, 70, 40, 153, 171, 145, 152, 212,
139, 237, 69, 74, 220, 152, 144, 198, 183, 40, 151, 123, 226, 246,
        104, 166, 81, 169, 184, 155, 140, 66, 189, 182, 165, 162, 57, 50, 136, 105, 133, 230, 53, 38, 102, 167, 113, 94, 254, 184, 211,
157, 177, 78, 116, 91, 125, 59, 45, 120, 106, 172, 193, 141, 247, 98, 125, 173, 6, 232, 168, 161, 126, 46, 197, 35, 157, 88, 222, 106, 101,
173, 7, 145, 213, 81, 139, 10, 133, 9, 108, 152, 205, 214, 135, 16, 136, 99, 58, 27, 45, 180, 14, 105, 107, 235, 180, 224, 98, 43, 30, 114,
222, 86, 27, 32, 183, 244, 89, 139, 214, 186, 74, 225, 201, 238, 187, 240, 198, 43, 111, 166, 54, 206, 107, 239, 189, 248, 206, 164, 86, 190,
252, 246, 235, 111, 131, 245, 254, 43, 240, 192, 242, 150, 75, 240, 193, 19, 5, 4, 0, 59, };
        BigGIFHelper(theClient, petaflopsplanner, 1418);
    }
    else if (stricmp(File, "logo.gif") == 0) {
        static unsigned char logo[1780] = { 71, 73, 70, 56, 57, 97, 151, 0, 60, 0, 179, 255, 0, 192, 192, 192, 204, 255, 255, 204, 204, 255,
204, 204, 204, 153, 204, 255, 153, 204, 204, 153, 153, 153, 102, 204, 204, 102, 153, 204, 102, 102, 102, 51, 153, 204, 51, 51, 51, 0, 153, 204,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 33, 249, 4, 1, 0, 0, 0, 0, 44, 0, 0, 0, 0, 151, 0, 60, 0, 0, 4, 255, 16, 200, 73, 171, 189, 56, 235, 205, 187, 255,
96, 248, 5, 129, 96, 18, 104, 170, 174, 132, 41, 144, 129, 40, 207, 116, 13, 18, 197, 129, 32, 74, 239, 255, 192, 160, 240, 183, 67, 28, 10,
49, 217, 96, 105, 107, 58, 55, 1, 222, 112, 74, 173, 246, 24, 10, 66, 104, 144, 104, 120, 19, 3, 219, 192, 96, 0, 112, 203, 79, 27, 193, 202,
110, 11, 17, 73, 206, 192, 219, 88, 208, 195, 52, 131, 23, 160, 95, 164, 107, 5, 110, 130, 131, 10, 113, 26, 122, 13, 97, 115, 13, 104, 51,
139, 102, 11, 9, 127, 51, 107, 132, 150, 86, 8, 29, 93, 126, 18, 118, 146, 52, 143, 147, 52, 1, 152, 70, 71, 5, 168, 44, 44, 168, 168, 7, 58,
82, 84, 5, 28, 155, 19, 6, 9, 104, 182, 9, 96, 18, 100, 185, 187, 188, 186, 136, 102, 183, 192, 186, 120, 162, 24, 129, 66, 5, 47, 162, 36, 4,
7, 67, 179, 94, 199,
        181, 116, 117, 157, 215, 117, 97, 93, 218, 124, 216, 221, 119, 200, 24, 66, 7, 227, 22, 149, 64, 90, 26, 139, 11, 6, 213, 158,
3, 158, 0, 118, 11, 99, 94, 239, 248, 92, 123, 125, 139, 239, 180, 231, 40, 164, 243, 177, 46, 3, 9, 23, 170, 76, 24, 218, 0, 171, 71, 166, 13,
136, 240, 73, 144, 103, 239, 155, 159, 121, 0, 186, 220, 194, 246, 173, 129, 69, 138, 97, 255, 16, 85, 59, 23, 13, 8, 20, 54, 12, 100, 121, 24,
216, 163, 131, 60, 58, 159, 40, 244, 161, 215, 224, 147, 198, 153, 29, 45, 86, 16, 25, 112, 66, 67, 5, 230, 52, 144, 98, 163, 178, 195, 208,
31, 2, 60, 236, 99, 52, 204, 142, 151, 139, 53, 37, 220, 228, 40, 108, 38, 23, 167, 212, 122, 74, 8, 82, 20, 131, 128, 54, 93, 57, 4, 73, 250,
161, 94, 198, 47, 180, 48, 78, 229, 84, 21, 155, 83, 93, 89, 181, 6, 41, 232, 21, 44, 136, 185, 25, 94, 54, 66, 196, 115, 166, 218, 154, 56,
219, 218, 203, 202, 179, 231, 81, 31, 100, 13, 146, 0, 240, 117, 74, 80, 15, 63, 195, 82, 88, 20, 179, 79, 96, 183, 81, 207, 110, 100, 203,
175, 142, 176, 142, 35, 145, 29, 238, 177, 80, 67, 227, 33, 143, 59, 148, 244, 33, 153, 130, 217, 108, 238, 56, 154, 253, 187, 49, 81, 78, 203,
89, 187, 133, 118, 22,
        164, 116, 134, 211, 111, 64, 172, 238, 145, 218, 2, 162, 72, 78, 203, 212, 115, 10, 213, 102, 212, 122, 9, 156, 126, 124, 138,
117, 247, 164, 209, 10, 64, 0, 15, 242, 208, 131, 50, 31, 221, 49, 24, 192, 202, 180, 163, 187, 5, 126, 186, 160, 209, 8, 9, 31, 122, 157, 199,
199, 87, 52, 28, 68, 160, 42, 21, 195, 133, 220, 87, 17, 231, 187, 67, 151, 239, 84, 192, 196, 255, 7, 214, 77, 84, 224, 56, 163, 133, 119,
137, 27, 12, 20, 196, 82, 120, 90, 69, 8, 66, 130, 20, 44, 232, 134, 131, 64, 64, 40, 225, 134, 39, 17, 81, 161, 133, 68, 77, 176, 93, 118, 24,
112, 97, 204, 68, 39, 106, 242, 75, 19, 41, 58, 49, 218, 99, 216, 129, 184, 140, 136, 245, 149, 8, 211, 68, 113, 109, 224, 143, 19, 59, 62,
241, 226, 4, 49, 202, 8, 68, 81, 35, 230, 69, 7, 39, 139, 20, 120, 6, 36, 243, 137, 33, 145, 143, 64, 192, 40, 100, 21, 68, 214, 120, 193, 34,
18, 37, 153, 129, 29, 141, 164, 209, 163, 139, 81, 2, 57, 101, 44, 52, 154, 100, 99, 29, 81, 105, 105, 6, 25, 120, 232, 165, 72, 53, 75, 28,
195, 4, 155, 147, 145, 49, 193, 151, 77, 252, 40, 65, 144, 99, 2, 37, 166, 153, 87, 226, 179, 135, 150, 228, 129, 161, 77, 95, 215, 124, 242,
212, 141, 47, 49, 250, 36,
        152, 63, 60, 54, 34, 105, 48, 252, 52, 23, 12, 254, 249, 32, 37, 160, 22, 36, 169, 15, 53, 34, 185, 233, 73, 62, 137, 248, 163,
23, 0, 79, 129, 182, 201, 61, 165, 62, 154, 103, 152, 18, 76, 26, 158, 165, 64, 36, 198, 18, 113, 127, 254, 96, 100, 34, 241, 128, 202, 17,
123, 92, 154, 225, 43, 103, 80, 161, 129, 143, 122, 194, 50, 229, 170, 13, 122, 50, 22, 156, 255, 79, 85, 216, 90, 78, 174, 62, 236, 186, 143,
161, 182, 81, 0, 44, 83, 73, 178, 167, 83, 176, 168, 150, 135, 163, 178, 226, 190, 26, 41, 181, 224, 81, 64, 43, 82, 19, 220, 234, 231, 158,
86, 118, 58, 172, 150, 87, 221, 24, 108, 183, 153, 249, 85, 94, 150, 209, 209, 81, 198, 178, 53, 52, 27, 227, 172, 85, 20, 52, 233, 166, 186,
158, 185, 132, 160, 173, 122, 54, 207, 189, 212, 120, 171, 175, 177, 140, 32, 2, 23, 185, 93, 50, 11, 43, 0, 3, 127, 72, 133, 193, 51, 194,
203, 169, 128, 89, 121, 50, 44, 47, 207, 113, 27, 113, 190, 152, 81, 60, 94, 121, 193, 2, 60, 202, 198, 216, 49, 160, 96, 193, 34, 98, 49, 100,
153, 9, 7, 106, 91, 56, 137, 120, 187, 109, 25, 73, 234, 241, 201, 196, 18, 184, 167, 50, 198, 105, 8, 28, 101, 1, 116, 89, 81, 80, 0, 168,
236, 28, 107, 188, 36, 219, 134, 101,
        34, 70, 103, 35, 9, 196, 92, 83, 149, 178, 203, 96, 255, 91, 174, 198, 231, 138, 76, 144, 5, 82, 179, 253, 67, 149, 35, 79, 22,
87, 117, 22, 51, 119, 150, 59, 161, 126, 193, 23, 77, 46, 111, 210, 47, 211, 80, 166, 189, 213, 15, 116, 13, 78, 102, 5, 86, 59, 27, 247, 157,
232, 225, 33, 95, 69, 221, 232, 18, 73, 83, 140, 52, 222, 158, 103, 217, 160, 193, 229, 255, 82, 209, 185, 67, 79, 147, 230, 166, 59, 129, 58,
21, 240, 201, 154, 219, 107, 95, 189, 56, 129, 7, 118, 218, 250, 157, 175, 163, 45, 186, 225, 61, 132, 101, 122, 237, 168, 247, 128, 225, 234,
28, 246, 78, 1, 133, 208, 158, 254, 187, 21, 182, 215, 218, 110, 134, 190, 39, 143, 1, 118, 234, 190, 93, 58, 241, 207, 179, 43, 65, 166, 26,
78, 246, 122, 156, 202, 195, 30, 2, 243, 62, 233, 252, 46, 186, 211, 82, 0, 28, 3, 137, 101, 90, 156, 220, 103, 91, 224, 173, 242, 56, 141,
208, 219, 4, 254, 21, 119, 251, 247, 170, 43, 176, 67, 28, 249, 157, 127, 167, 204, 218, 102, 150, 134, 46, 31, 104, 159, 7, 176, 227, 155,
197, 12, 175, 10, 231, 243, 13, 0, 34, 179, 171, 140, 169, 207, 127, 78, 232, 26, 235, 182, 55, 22, 15, 76, 106, 99, 28, 96, 224, 153, 50, 54,
134, 70, 176, 103, 64, 119, 10, 144, 129, 222,
        180, 38, 17, 238, 163, 73, 99, 144, 19, 19, 72, 56, 17, 17, 214, 194, 133, 0, 192, 75, 7, 46, 40, 56, 177, 144, 110, 131, 140,
59, 18, 55, 234, 240, 150, 9, 132, 3, 114, 53, 233, 2, 171, 62, 117, 141, 48, 96, 165, 34, 70, 139, 142, 85, 176, 178, 139, 70, 109, 67, 2, 26,
228, 0, 13, 53, 53, 192, 10, 226, 208, 107, 39, 60, 11, 96, 130, 101, 25, 145, 255, 68, 164, 54, 100, 224, 146, 45, 48, 119, 19, 90, 68, 132,
12, 152, 185, 197, 170, 196, 136, 145, 159, 84, 239, 2, 2, 240, 158, 16, 108, 102, 193, 247, 93, 113, 71, 69, 19, 91, 122, 202, 163, 30, 158,
0, 132, 61, 93, 163, 151, 175, 152, 210, 143, 172, 248, 99, 15, 119, 155, 158, 29, 161, 224, 138, 34, 56, 178, 145, 173, 185, 64, 166, 72, 116,
69, 51, 180, 201, 87, 71, 107, 153, 84, 0, 195, 17, 236, 105, 230, 27, 146, 80, 147, 29, 238, 65, 44, 65, 50, 34, 75, 34, 28, 209, 27, 199,
113, 43, 253, 237, 175, 92, 78, 12, 27, 177, 248, 182, 201, 205, 104, 139, 137, 160, 76, 22, 30, 70, 121, 153, 216, 4, 81, 55, 17, 25, 201, 27,
20, 56, 137, 73, 102, 129, 29, 143, 138, 136, 83, 72, 137, 178, 98, 213, 18, 39, 141, 234, 97, 32, 231, 86, 49, 61, 70, 164, 136, 29, 201, 150,
4, 220, 101, 63,
        87, 164, 162, 5, 47, 128, 1, 20, 96, 112, 2, 28, 228, 96, 93, 171, 28, 23, 26, 240, 120, 178, 111, 241, 145, 147, 156, 88, 213,
39, 167, 153, 45, 94, 234, 209, 84, 158, 28, 198, 163, 230, 215, 167, 42, 164, 68, 71, 79, 50, 229, 61, 50, 233, 76, 205, 224, 68, 104, 81,
161, 231, 46, 101, 217, 76, 53, 177, 201, 78, 52, 225, 196, 54, 251, 185, 32, 87, 162, 111, 161, 157, 132, 105, 103, 215, 144, 117, 22, 52,
198, 51, 51, 243, 144, 160, 33, 53, 186, 141, 140, 214, 228, 87, 28, 1, 31, 69, 253, 25, 201, 172, 193, 4, 58, 119, 232, 134, 47, 203, 192, 23,
96, 162, 180, 38, 210, 252, 194, 9, 95, 230, 135, 94, 210, 228, 22, 245, 160, 204, 203, 98, 194, 179, 149, 90, 193, 162, 46, 197, 135, 50, 87,
134, 158, 84, 101, 195, 95, 58, 185, 28, 154, 62, 177, 76, 154, 64, 213, 167, 78, 68, 226, 53, 64, 39, 144, 117, 25, 85, 1, 72, 0, 65, 156,
176, 151, 66, 75, 154, 245, 172, 33, 52, 33, 8, 215, 52, 86, 237, 181, 16, 132, 112, 149, 19, 12, 151, 0, 195, 229, 9, 0, 7, 175, 152, 210, 14,
78, 65, 0, 98, 102, 239, 175, 51, 128, 65, 9, 92, 64, 216, 194, 134, 211, 128, 128, 77, 108, 7, 34, 0, 0, 59, };
        BigGIFHelper(theClient, logo, 1780);
    }
    else if (stricmp(File, "pimchip.gif") == 0) {
        static unsigned char pimchip[1277] = { 71, 73, 70, 56, 55, 97, 83, 0, 54, 0, 247, 0, 0, 0, 0, 0, 128, 0, 0, 0, 128, 0, 128, 128, 0, 0,
0, 128, 128, 0, 128, 0, 128, 128, 192, 192, 192, 192, 220, 192, 166, 202, 240, 0, 0, 0, 0, 0, 42, 0, 0, 85, 0, 0, 127, 0, 0, 170, 0, 0, 212, 0,
42, 0, 0, 42, 42, 0, 42, 85, 0, 42, 127, 0, 42, 170, 0, 42, 212, 0, 85, 0, 0, 85, 42, 0, 85, 85, 0, 85, 127, 0, 85, 170, 0, 85, 212, 0, 127, 0,
0, 127, 42, 0, 127, 85, 0, 127, 127, 0, 127, 170, 0, 127, 212, 0, 170, 0, 0, 170, 42, 0, 170, 85, 0, 170, 127, 0, 170, 170, 0, 170, 212, 0,
212, 0, 0, 212, 42, 0, 212, 85, 0, 212, 127, 0, 212, 170, 0, 212, 212, 42, 0, 0, 42, 0, 42, 42, 0, 85, 42, 0, 127, 42, 0, 170, 42, 0, 212, 42,
42, 0, 42, 42, 42, 42, 42, 42, 85, 42, 42, 127, 42, 42, 170, 42, 42, 212, 42, 85, 0, 42, 85, 42, 42, 85, 85, 42, 85, 127, 42, 85, 170, 42, 85, 212,
42, 127, 0, 42, 127, 42, 42, 127, 85, 42, 127, 127, 42, 127, 170, 42, 127, 212, 42, 170, 0, 42, 170, 42, 42, 170, 85, 42, 170, 127, 42, 170,
170, 42, 170, 212, 42, 212, 0, 42, 212, 42, 42, 212, 85, 42, 212, 127, 42, 212, 170,
        42, 212, 212, 85, 0, 0, 85, 0, 42, 85, 0, 85, 85, 0, 127, 85, 0, 170, 85, 0, 212, 85, 42, 0, 85, 42, 42, 85, 42, 85, 85, 42,
127, 85, 42, 170, 85, 42, 212, 85, 85, 0, 85, 85, 42, 85, 85, 85, 85, 85, 127, 85, 85, 170, 85, 85, 212, 85, 127, 0, 85, 127, 42, 85, 127, 85,
85, 127, 127, 85, 127, 170, 85, 127, 212, 85, 170, 0, 85, 170, 42, 85, 170, 85, 85, 170, 127, 85, 170, 170, 85, 170, 212, 85, 212, 0, 85, 212,
42, 85, 212, 85, 85, 212, 127, 85, 212, 170, 85, 212, 212, 127, 0, 0, 127, 0, 42, 127, 0, 85, 127, 0, 127, 127, 0, 170, 127, 0, 212, 127, 42,
0, 127, 42, 42, 127, 42, 85, 127, 42, 127, 127, 42, 170, 127, 42, 212, 127, 85, 0, 127, 85, 42, 127, 85, 85, 127, 85, 127, 127, 85, 170, 127,
85, 212, 127, 127, 0, 127, 127, 42, 127, 127, 85, 127, 127, 127, 127, 170, 127, 127, 212, 127, 170, 0, 127, 170, 42, 127, 170, 85, 127,
170, 127, 127, 170, 170, 127, 170, 212, 127, 212, 0, 127, 212, 42, 127, 212, 85, 127, 212, 127, 127, 212, 170, 127, 212, 212, 170, 0, 0, 170,
0, 42, 170, 0, 85, 170, 0, 127, 170, 0, 170, 170, 0, 212, 170, 42, 0, 170, 42, 42, 170, 42, 85, 170, 42, 127, 170, 42, 170, 170, 42, 212, 170,
        85, 0, 170, 85, 42, 170, 85, 85, 170, 85, 127, 170, 85, 170, 170, 85, 212, 170, 127, 0, 170, 127, 42, 170, 127, 85, 170, 127,
127, 170, 127, 170, 170, 127, 212, 170, 170, 0, 170, 170, 42, 170, 170, 85, 170, 170, 127, 170, 170, 170, 170, 170, 212, 170, 212, 0, 170, 212,
42, 170, 212, 85, 170, 212, 127, 170, 212, 170, 212, 212, 212, 0, 0, 212, 0, 42, 212, 0, 85, 212, 0, 127, 212, 0, 170, 212, 0, 212, 212,
42, 0, 212, 42, 42, 212, 42, 85, 212, 42, 127, 212, 42, 170, 212, 42, 212, 212, 85, 0, 212, 85, 42, 212, 85, 85, 212, 85, 127, 212, 85, 170,
212, 85, 212, 212, 127, 0, 212, 127, 42, 212, 127, 85, 212, 127, 127, 212, 127, 170, 212, 127, 212, 212, 170, 0, 212, 170, 42, 212, 170, 85,
212, 170, 127, 212, 170, 170, 212, 170, 212, 212, 212, 0, 212, 212, 42, 212, 212, 85, 212, 212, 127, 212, 212, 170, 212, 212, 212, 0, 0, 0, 12,
12, 12, 25, 25, 25, 38, 38, 38, 51, 51, 51, 63, 63, 63, 76, 76, 76, 89, 89, 89, 102, 102, 102, 114, 114, 114, 127, 127, 127, 140, 140, 140,
153, 153, 153, 165, 165, 165, 178, 178, 178, 191, 191, 191, 204, 204, 204, 216, 216, 216, 229, 229, 229, 242, 242, 242, 255, 251, 240, 160,
160, 164, 128, 128, 128, 255, 0, 0, 0, 255, 0, 255, 255,
        0, 0, 0, 255, 255, 0, 255, 0, 255, 255, 255, 255, 255, 44, 0, 0, 0, 0, 83, 0, 54, 0, 0, 8, 254, 0, 1, 8, 252, 71, 144, 160, 64,
0, 5, 13, 14, 76, 120, 48, 225, 191, 134, 12, 23, 22, 132, 56, 81, 162, 66, 132, 17, 49, 86, 60, 200, 177, 163, 199, 143, 32, 67, 138, 28, 73,
```

```
114, 228, 190, 147, 40, 83, 170, 92, 201, 178, 165, 203, 151, 48, 99, 238, 27, 8, 64, 166, 205, 155, 56, 115, 166, 196, 88, 115, 102, 201, 159,
34, 85, 2, 5, 186, 211, 224, 201, 158, 58, 113, 34, 61, 154, 212, 229, 82, 158, 76, 155, 222, 92, 58, 83, 42, 203, 167, 8, 123, 82, 181, 234,
84, 40, 87, 175, 40, 57, 70, 253, 250, 146, 234, 86, 171, 88, 5, 142, 37, 219, 210, 44, 219, 181, 89, 181, 30, 21, 43, 243, 236, 85, 176, 97,
213, 206, 213, 91, 53, 236, 206, 162, 15, 163, 62, 173, 107, 211, 173, 95, 193, 135, 7, 31, 62, 106, 180, 175, 226, 131, 99, 233, 66, 174, 138,
212, 112, 95, 188, 114, 23, 95, 158, 217, 88, 171, 100, 191, 149, 29, 131, 86, 188, 248, 108, 232, 205, 164, 57, 7, 22, 125, 57, 180, 235, 189,
106, 45, 71, 190, 58, 25, 245, 223, 188, 171, 51, 179, 166, 12, 122, 237, 230, 223, 167, 241, 106, 78, 253, 185, 245, 108, 207, 187, 107, 202,
182, 141, 88,
                56, 241, 201, 186, 29, 243, 141, 237, 115, 175, 239, 229, 181, 77, 215, 134, 251, 48, 250, 91, 225, 191, 185, 227, 62, 237,
252, 189, 45, 120, 241, 128, 189, 151, 191, 254, 125, 124, 238, 245, 43, 151, 163, 15, 75, 30, 254, 109, 205, 95, 177, 230, 30, 202, 159, 110,
222, 254, 65, 253, 103, 151, 125, 4, 198, 183, 93, 129, 8, 198, 215, 93, 130, 12, 42, 184, 90, 131, 16, 66, 5, 97, 132, 141, 77, 200, 160, 132,
22, 38, 72, 83, 134, 0, 242, 231, 19, 95, 17, 22, 168, 220, 129, 23, 138, 200, 25, 136, 13, 14, 72, 150, 114, 21, 134, 72, 32, 139, 15, 186,
232, 95, 71, 206, 97, 166, 154, 138, 246, 233, 22, 221, 107, 115, 229, 5, 90, 139, 41, 194, 197, 28, 111, 61, 22, 73, 89, 86, 28, 10, 73, 99,
111, 76, 41, 103, 156, 127, 50, 254, 181, 99, 98, 188, 185, 70, 162, 134, 74, 150, 246, 153, 147, 174, 45, 104, 161, 142, 179, 1, 199, 90, 151,
49, 6, 249, 97, 112, 68, 50, 9, 38, 134, 46, 230, 120, 99, 146, 47, 190, 249, 165, 137, 27, 206, 25, 39, 148, 102, 186, 137, 231, 133, 29, 18,
21, 87, 134, 23, 2, 9, 40, 124, 60, 245, 105, 232, 161, 63, 57, 212, 157, 70, 23, 57, 68, 81, 163, 25, 57, 106, 209, 162, 146, 50, 74, 105,
164, 4, 5, 4, 0, 59, };
        BigGIFHelper(theClient, pimchip, 1277);
    }
    else if (stricmp(File, "redchip.gif") == 0) {
        static unsigned char redchip[1246] = { 71, 73, 70, 56, 55, 97, 78, 0, 48, 0, 247, 0, 0, 0, 0, 0, 128, 0, 0, 0, 128, 0, 128, 128, 0, 0,
0, 128, 128, 0, 128, 0, 128, 128, 192, 192, 192, 192, 220, 192, 166, 202, 240, 0, 0, 0, 0, 0, 42, 0, 0, 0, 85, 0, 0, 127, 0, 0, 170, 0, 0, 212, 0,
42, 0, 0, 42, 42, 0, 42, 85, 0, 42, 127, 0, 42, 170, 0, 42, 212, 0, 85, 0, 0, 85, 42, 0, 85, 85, 0, 85, 127, 0, 85, 170, 0, 85, 212, 0, 127, 0,
0, 127, 42, 0, 127, 85, 0, 127, 127, 0, 127, 170, 0, 127, 212, 0, 170, 0, 0, 170, 42, 0, 170, 85, 0, 170, 127, 0, 170, 170, 0, 170, 212, 0,
212, 0, 0, 212, 42, 0, 212, 85, 0, 212, 127, 0, 212, 170, 0, 212, 212, 42, 0, 0, 42, 0, 42, 42, 0, 85, 42, 0, 127, 42, 0, 170, 42, 0, 212, 42,
42, 0, 42, 42, 42, 42, 42, 85, 42, 42, 127, 42, 42, 170, 42, 42, 212, 42, 85, 0, 42, 85, 42, 42, 85, 85, 42, 85, 127, 42, 85, 170, 42, 85, 212,
42, 127, 0, 42, 127, 42, 42, 127, 85, 42, 127, 127, 42, 127, 170, 42, 127, 212, 42, 170, 0, 42, 170, 42, 42, 170, 85, 42, 170, 127, 42, 170,
170, 42, 170, 212, 42, 212, 0, 42, 212, 42, 42, 212, 85, 42, 212, 127, 42, 212, 170,
                42, 212, 212, 85, 0, 0, 85, 0, 42, 85, 0, 85, 85, 0, 127, 85, 0, 170, 85, 0, 212, 85, 42, 0, 85, 42, 42, 85, 42, 85, 85, 42,
127, 85, 42, 170, 85, 42, 212, 85, 85, 0, 85, 85, 42, 85, 85, 85, 85, 85, 127, 85, 85, 170, 85, 85, 212, 85, 127, 0, 85, 127, 42, 85, 127, 85,
85, 127, 127, 85, 127, 170, 85, 127, 212, 85, 170, 0, 85, 170, 42, 85, 170, 85, 85, 170, 127, 85, 170, 170, 85, 170, 212, 85, 212, 0, 85, 212,
42, 85, 212, 85, 85, 212, 127, 85, 212, 170, 85, 212, 212, 127, 0, 0, 127, 0, 42, 127, 0, 85, 127, 0, 127, 127, 0, 170, 127, 0, 212, 127, 42,
0, 127, 42, 42, 127, 42, 85, 127, 42, 127, 127, 42, 170, 127, 42, 212, 127, 85, 0, 127, 85, 42, 127, 85, 85, 127, 85, 127, 85, 170, 127,
85, 212, 127, 127, 0, 127, 127, 42, 127, 127, 85, 127, 127, 127, 127, 127, 170, 127, 127, 212, 127, 170, 0, 127, 170, 42, 127, 170, 85, 127,
170, 127, 127, 170, 170, 127, 170, 212, 127, 212, 0, 127, 212, 42, 127, 212, 85, 127, 212, 127, 127, 212, 170, 127, 212, 212, 170, 0, 0, 170,
0, 42, 170, 0, 85, 170, 0, 127, 170, 0, 170, 170, 0, 212, 170, 42, 0, 170, 42, 42, 170, 42, 85, 170, 42, 127, 170, 42, 170, 170, 42, 212, 170,
                85, 0, 170, 85, 42, 170, 85, 85, 170, 85, 127, 170, 85, 170, 170, 85, 212, 170, 127, 0, 170, 127, 42, 170, 127, 85, 170, 127,
127, 170, 127, 170, 170, 127, 212, 170, 170, 0, 170, 170, 42, 170, 170, 85, 170, 170, 127, 170, 170, 170, 170, 170, 212, 170, 212, 0, 170, 212,
42, 170, 212, 85, 170, 212, 127, 170, 212, 170, 170, 212, 212, 212, 0, 0, 212, 0, 42, 212, 0, 85, 212, 0, 127, 212, 0, 170, 212, 0, 212, 212,
42, 0, 212, 42, 42, 212, 42, 85, 212, 42, 127, 212, 42, 170, 212, 42, 212, 212, 85, 0, 212, 85, 42, 212, 85, 85, 212, 85, 127, 212, 85, 170,
212, 85, 212, 212, 127, 0, 212, 127, 42, 212, 127, 85, 212, 127, 127, 212, 127, 170, 212, 127, 212, 212, 170, 0, 212, 170, 42, 212, 170, 85,
212, 170, 127, 212, 170, 170, 212, 170, 212, 212, 212, 0, 212, 212, 42, 212, 212, 85, 212, 212, 127, 212, 212, 170, 0, 212, 212, 212, 0, 0, 0, 12,
12, 12, 25, 25, 25, 38, 38, 38, 51, 51, 51, 63, 63, 63, 76, 76, 76, 89, 89, 89, 102, 102, 102, 114, 114, 114, 127, 127, 127, 140, 140, 140,
153, 153, 153, 165, 165, 165, 178, 178, 178, 191, 191, 191, 204, 204, 204, 216, 216, 216, 229, 229, 229, 242, 242, 242, 255, 251, 240, 160,
160, 164, 128, 128, 128, 255, 0, 0, 0, 255, 0, 255, 255,
                0, 0, 0, 255, 255, 0, 255, 0, 255, 255, 255, 255, 255, 44, 0, 0, 0, 0, 78, 0, 48, 0, 0, 8, 254, 0, 255, 9, 20, 8, 160, 224, 64,
130, 6, 15, 22, 4, 112, 240, 223, 194, 134, 15, 27, 34, 100, 168, 48, 225, 192, 136, 23, 45, 58, 92, 200, 177, 163, 199, 143, 32, 59, 86, 12,
73, 146, 164, 194, 124, 40, 83, 170, 92, 201, 178, 165, 75, 141, 0, 92, 202, 156, 153, 207, 98, 65, 154, 56, 115, 82, 68, 152, 179, 103, 205,
138, 62, 131, 174, 220, 185, 81, 40, 77, 162, 55, 141, 10, 69, 26, 83, 233, 75, 136, 78, 131, 18, 117, 24, 181, 229, 212, 166, 29, 123, 54,
197, 121, 117, 104, 210, 154, 11, 83, 110, 173, 169, 178, 43, 89, 177, 58, 181, 34, 45, 139, 246, 236, 217, 177, 112, 215, 186, 125, 155, 116,
43, 71, 148, 97, 201, 218, 149, 59, 183, 237, 91, 182, 98, 229, 102, 117, 27, 211, 238, 95, 188, 96, 203, 242, 29, 235, 23, 43, 96, 188, 59,
235, 34, 158, 172, 23, 45, 214, 187, 140, 153, 250, 85, 252, 21, 110, 89, 160, 132, 41, 23, 182, 76, 185, 244, 207, 140, 155, 83, 207, 133, 11,
218, 112, 232, 203, 135, 43, 127, 70, 221, 216, 244, 234, 217, 23, 75, 95, 222, 171, 119, 183, 105, 179, 129, 93, 123, 253, 138, 23, 106, 85,
157, 198, 143, 139, 77, 174, 92,
                38, 204, 230, 129, 79, 66, 119, 206, 183, 249, 243, 233, 47, 171, 43, 183, 201, 24, 251, 114, 218, 208, 153, 196, 150, 28, 79,
94, 124, 249, 242, 35, 33, 106, 220, 120, 117, 61, 70, 245, 237, 227, 195, 151, 72, 191, 190, 253, 251, 248, 243, 235, 223, 207, 191, 191, 255,
255, 0, 6, 40, 224, 128, 4, 22, 104, 224, 129, 8, 38, 168, 160, 128, 239, 177, 55, 95, 122, 11, 254, 119, 222, 132, 34, 69, 120, 95, 119, 216,
173, 103, 161, 116, 222, 69, 183, 33, 125, 24, 78, 167, 225, 135, 196, 117, 56, 213, 135, 60, 129, 213, 217, 93, 143, 245, 101, 23, 138, 18,
73, 54, 153, 103, 136, 241, 6, 217, 119, 48, 74, 231, 25, 141, 178, 37, 118, 99, 113, 57, 234, 200, 214, 96, 53, 146, 54, 26, 144, 65, 18, 164,
91, 108, 161, 85, 118, 36, 88, 73, 42, 121, 155, 112, 131, 141, 246, 228, 136, 36, 138, 166, 165, 139, 135, 237, 21, 101, 93, 121, 77, 57, 84,
151, 56, 6, 89, 162, 119, 39, 162, 120, 102, 134, 81, 74, 217, 33, 146, 73, 134, 24, 94, 155, 84, 189, 9, 89, 155, 20, 230, 137, 165, 154, 238,
245, 41, 223, 72, 1, 1, 0, 59, };
        BigGIFHelper(theClient, redchip, 1246);
    }
    else if (stricmp(File, "ncube.gif") == 0) {
        static unsigned char ncube[1287] = { 71, 73, 70, 56, 55, 97, 81, 0, 51, 0, 247, 0, 0, 0, 0, 0, 128, 0, 0, 0, 128, 0, 128, 128, 0, 0, 0,
128, 128, 0, 128, 0, 128, 128, 192, 192, 192, 192, 220, 192, 166, 202, 240, 0, 0, 0, 0, 0, 42, 0, 0, 0, 85, 0, 0, 127, 0, 0, 170, 0, 0, 212, 0,
42, 0, 0, 42, 42, 0, 42, 85, 0, 42, 127, 0, 42, 170, 0, 42, 212, 0, 85, 0, 0, 85, 42, 0, 85, 85, 0, 85, 127, 0, 85, 170, 0, 85, 212, 0, 127, 0,
0, 127, 42, 0, 127, 85, 0, 127, 127, 0, 127, 170, 0, 127, 212, 0, 170, 0, 0, 170, 42, 0, 170, 85, 0, 170, 127, 0, 170, 170, 0, 170, 212, 0,
212, 0, 0, 212, 42, 0, 212, 85, 0, 212, 127, 0, 212, 170, 0, 212, 212, 42, 0, 0, 42, 0, 42, 42, 0, 85, 42, 0, 127, 42, 0, 170, 42, 0, 212, 42,
42, 0, 42, 42, 42, 42, 42, 85, 42, 42, 127, 42, 42, 170, 42, 42, 212, 42, 85, 0, 42, 85, 42, 42, 85, 85, 42, 85, 127, 42, 85, 170, 42, 85, 212,
42, 127, 0, 42, 127, 42, 42, 127, 85, 42, 127, 127, 42, 127, 170, 42, 127, 212, 42, 170, 0, 42, 170, 42, 42, 170, 85, 42, 170, 127, 42, 170,
170, 42, 170, 212, 42, 212, 0, 42, 212, 42, 42, 212, 85, 42, 212, 127, 42, 212, 170,
                42, 212, 212, 85, 0, 0, 85, 0, 42, 85, 0, 85, 85, 0, 127, 85, 0, 170, 85, 0, 212, 85, 42, 0, 85, 42, 42, 85, 42, 85, 85, 42,
127, 85, 42, 170, 85, 42, 212, 85, 85, 0, 85, 85, 42, 85, 85, 85, 85, 85, 127, 85, 85, 170, 85, 85, 212, 85, 127, 0, 85, 127, 42, 85, 127, 85,
85, 127, 127, 85, 127, 170, 85, 127, 212, 85, 170, 0, 85, 170, 42, 85, 170, 85, 85, 170, 127, 85, 170, 170, 85, 170, 212, 85, 212, 0, 85, 212,
42, 85, 212, 85, 85, 212, 127, 85, 212, 170, 85, 212, 212, 127, 0, 0, 127, 0, 42, 127, 0, 85, 127, 0, 127, 127, 0, 170, 127, 0, 212, 127, 42,
0, 127, 42, 42, 127, 42, 85, 127, 42, 127, 127, 42, 170, 127, 42, 212, 127, 85, 0, 127, 85, 42, 127, 85, 85, 127, 85, 127, 85, 170, 127,
85, 212, 127, 127, 0, 127, 127, 42, 127, 127, 85, 127, 127, 127, 127, 127, 170, 127, 127, 212, 127, 170, 0, 127, 170, 42, 127, 170, 85, 127,
170, 127, 127, 170, 170, 127, 170, 212, 127, 212, 0, 127, 212, 42, 127, 212, 85, 127, 212, 127, 127, 212, 170, 127, 212, 212, 170, 0, 0, 170,
0, 42, 170, 0, 85, 170, 0, 127, 170, 0, 170, 170, 0, 212, 170, 42, 0, 170, 42, 42, 170, 42, 85, 170, 42, 127, 170, 42, 170, 170, 42, 212, 170,
                85, 0, 170, 85, 42, 170, 85, 85, 170, 85, 127, 170, 85, 170, 170, 85, 212, 170, 127, 0, 170, 127, 42, 170, 127, 85, 170, 127,
127, 170, 127, 170, 170, 127, 212, 170, 170, 0, 170, 170, 42, 170, 170, 85, 170, 170, 127, 170, 170, 170, 170, 170, 212, 170, 212, 0, 170, 212,
42, 170, 212, 85, 170, 212, 127, 170, 212, 170, 170, 212, 212, 212, 0, 0, 212, 0, 42, 212, 0, 85, 212, 0, 127, 212, 0, 170, 212, 0, 212, 212,
42, 0, 212, 42, 42, 212, 42, 85, 212, 42, 127, 212, 42, 170, 212, 42, 212, 212, 85, 0, 212, 85, 42, 212, 85, 85, 212, 85, 127, 212, 85, 170,
212, 85, 212, 212, 127, 0, 212, 127, 42, 212, 127, 85, 212, 127, 127, 212, 127, 170, 212, 127, 212, 212, 170, 0, 212, 170, 42, 212, 170, 85,
212, 170, 127, 212, 170, 170, 212, 170, 212, 212, 212, 0, 212, 212, 42, 212, 212, 85, 212, 212, 127, 212, 212, 170, 212, 212, 212, 0, 0, 0, 12,
12, 12, 25, 25, 25, 38, 38, 38, 51, 51, 51, 63, 63, 63, 76, 76, 76, 89, 89, 89, 102, 102, 102, 114, 114, 114, 127, 127, 127, 140, 140, 140,
153, 153, 153, 165, 165, 165, 178, 178, 178, 191, 191, 191, 204, 204, 204, 216, 216, 216, 229, 229, 229, 242, 242, 242, 255, 251, 240, 160,
160, 164, 128, 128, 128, 255, 0, 0, 0, 255, 0, 255, 255,
                0, 0, 0, 255, 255, 0, 255, 255, 255, 255, 255, 44, 0, 0, 0, 0, 81, 0, 51, 0, 0, 8, 254, 0, 255, 9, 28, 8, 160, 224, 64,
129, 5, 1, 28, 252, 151, 112, 97, 195, 131, 15, 9, 26, 132, 56, 81, 162, 66, 138, 23, 23, 34, 76, 200, 177, 163, 199, 143, 32, 67, 138, 28, 73,
178, 97, 193, 3, 40, 83, 170, 92, 201, 178, 165, 203, 151, 48, 99, 198, 84, 120, 82, 166, 205, 155, 56, 115, 178, 164, 9, 0, 101, 201, 159, 35,
87, 2, 5, 234, 147, 166, 79, 157, 72, 15, 244, 84, 185, 52, 105, 203, 165, 6, 161, 58, 205, 217, 244, 232, 212, 157, 69, 25, 90, 189, 42, 179,
170, 82, 174, 66, 125, 34, 220, 10, 246, 165, 87, 175, 92, 161, 142, 253, 90, 22, 230, 217, 182, 86, 121, 198, 229, 120, 19, 173, 203, 183, 76,
107, 42, 77, 152, 178, 170, 95, 177, 53, 165, 246, 181, 105, 247, 105, 216, 193, 113, 17, 255, 85, 204, 80, 240, 98, 190, 108, 247, 6, 14, 28,
25, 111, 100, 166, 138, 49, 43, 54, 202, 182, 227, 214, 158, 77, 65, 127, 222, 171, 25, 241, 101, 211, 130, 201, 66, 229, 236, 24, 117, 229,
206, 116, 45, 143, 222, 9, 249, 242, 99, 214, 179, 67, 191, 78, 77, 246, 180, 110, 172, 135, 85, 139, 213, 218, 57, 243, 209, 213, 197, 143, 7,
23, 158, 120,
                249, 227, 181, 188, 87, 135, 22, 61, 153, 183, 111, 161, 122, 237, 210, 213, 44, 247, 52, 220, 210, 189, 211, 237, 2, 182, 254,
221, 116, 248, 171, 106, 41, 151, 7, 110, 30, 236, 106, 226, 133, 191, 203, 118, 159, 149, 252, 250, 249, 226, 247, 194, 31, 202, 127, 123,
223, 254, 65, 41, 181, 214, 122, 4, 214, 181, 17, 128, 8, 38, 200, 159, 69, 14, 85, 116, 96, 131, 25, 61, 136, 17, 132, 20, 78, 104, 161, 70,
24, 102, 168, 225, 134, 28, 118, 232, 225, 135, 32, 134, 40, 226, 136, 36, 150, 104, 226, 137, 40, 166, 168, 81, 68, 18, 50, 120, 161, 138, 41,
42, 248, 19, 140, 36, 198, 87, 150, 131, 52, 122, 104, 163, 123, 17, 230, 216, 225, 142, 105, 245, 232, 163, 134, 122, 17, 40, 228, 144, 25,
86, 55, 93, 109, 204, 161, 134, 164, 142, 169, 69, 55, 216, 116, 255, 245, 245, 164, 142, 194, 45, 246, 21, 149, 69, 89, 121, 37, 135, 82, 122,
54, 229, 152, 162, 13, 247, 37, 145, 89, 54, 41, 21, 117, 255, 157, 137, 166, 109, 175, 21, 5, 25, 155, 211, 185, 153, 100, 110, 198, 141, 153,
156, 126, 118, 54, 40, 231, 111, 202, 97, 214, 154, 153, 125, 74, 84, 32, 161, 133, 54, 118, 168, 128, 137, 66, 180, 232, 145, 125, 2, 137, 94,
163, 142, 30, 138, 99, 161, 50, 206, 72, 169, 139, 156, 182, 232,
                105, 99, 0, 4, 4, 0, 59, };
        BigGIFHelper(theClient, ncube, 1287);
    }
    else if (stricmp(File, "bkgrnd.gif") == 0) {
        static unsigned char bkgrnd[83] = { 71, 73, 70, 56, 57, 97, 220, 5, 1, 0, 128, 255, 0, 255, 255, 255, 0, 51, 102, 44, 0, 0, 0, 0, 220,
5, 1, 0, 0, 2, 50, 140, 143, 169, 203, 237, 15, 163, 156, 148, 129, 139, 179, 222, 188, 251, 15, 134, 226, 72, 150, 230, 137, 166, 234, 202,
182, 238, 11, 199, 242, 76, 215, 246, 141, 231, 250, 206, 247, 254, 15, 12, 10, 135, 196, 162, 241, 136, 156, 21, 0, 0, 59, };
```

```
        BigGIFHelper(theClient, bkgrnd, 83);
    }
        else if (stricmp(File, "black.gif") == 0) {
                static unsigned char blackgif[4] =  { 16, 68, 0, 59, };
                SmallGIFHelper(theClient, blackgif);
        }
        else if (stricmp(File, "red.gif") == 0) {
                static unsigned char redgif[4] =    { 48, 69, 0, 59, };
                SmallGIFHelper(theClient, redgif);
        }
        else if (stricmp(File, "green.gif") == 0) {
                static unsigned char greengif[4] =  { 80, 69, 0, 59, };
                SmallGIFHelper(theClient, greengif);
        }
        else if (stricmp(File, "yellow.gif") == 0) {
                static unsigned char yellowgif[4] = { 112, 69, 0, 59, };
                SmallGIFHelper(theClient, yellowgif);    ,
        }
        else if (stricmp(File, "blue.gif") == 0) {
                static unsigned char bluegif[4] =   { 144, 69, 0, 59,  };
                SmallGIFHelper(theClient, bluegif);
        }
        else if (stricmp(File, "white.gif") == 0) {
                static unsigned char whitegif[4] =   { 240, 69, 0, 59,  };
                SmallGIFHelper(theClient, whitegif);
        }
        else if (stricmp(File, "dkyellow.gif") == 0) {
                static unsigned char dkyellowgif[4] =   { 112, 68, 0, 59, };
                SmallGIFHelper(theClient, dkyellowgif);
        }
    else if (stricmp(File, "w.gif") == 0) {
        static unsigned char w[92] = { 71, 73, 70, 56, 55, 97, 16, 0, 16, 0, 179, 0, 0, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 44, 0, 0, 0, 0, 16, 0, 16, 0, 0, 4, 17, 16, 200, 73, 171, 189, 56, 235, 205, 187, 255, 96, 40,
142, 100, 105, 78, 17, 0, 59, };
        BigGIFHelper(theClient, w, 92);
    }
    else if (stricmp(File, "a.gif") == 0) {
        static unsigned char a[113] = { 71, 73, 70, 56, 55, 97, 16, 0, 16, 0, 179, 0, 0, 0, 0, 0, 0, 206, 156, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 44, 0, 0, 0, 0, 16, 0, 16, 0, 0, 4, 38, 80, 200, 73, 171, 189, 19, 96, 171, 55, 237, 158, 4, 134, 163,
87, 110, 229, 41, 86, 64, 112, 165, 129, 203, 114, 241, 204, 6, 192, 168, 230, 118, 40, 168, 47, 159, 36, 2, 0, 59, };
        BigGIFHelper(theClient, a, 113);
    }
    else if (stricmp(File, "b.gif") == 0) {
        static unsigned char b[122] = { 71, 73, 70, 56, 55, 97, 16, 0, 16, 0, 179, 0, 0, 0, 0, 0, 0, 206, 156, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 44, 0, 0, 0, 0, 16, 0, 16, 0, 0, 4, 47, 16, 200, 73, 43, 16, 56, 235, 125, 183, 199, 221, 199, 105,
211, 23, 10, 64, 16, 156, 32, 169, 174, 163, 246, 178, 180, 74, 115, 118, 156, 165, 48, 233, 241, 186, 221, 234, 196, 2, 221, 68, 65, 36, 74,
169, 137, 0, 0, 59, };
        BigGIFHelper(theClient, b, 122);
    }
    else if (stricmp(File, "c.gif") == 0) {
        static unsigned char c[123] = { 71, 73, 70, 56, 55, 97, 16, 0, 16, 0, 179, 0, 0, 0, 0, 0, 0, 206, 156, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 44, 0, 0, 0, 0, 16, 0, 16, 0, 0, 4, 48, 16, 200, 73, 43, 16, 56, 235, 125, 183, 199, 221, 199, 105,
211, 23, 10, 64, 16, 156, 32, 169, 174, 163, 246, 178, 172, 160, 210, 31, 76, 138, 184, 24, 251, 25, 90, 13, 69, 74, 213, 78, 41, 221, 46, 56,
219, 68, 0, 0, 59, };
        BigGIFHelper(theClient, c, 123);
    }
    else if (stricmp(File, "d.gif") == 0) {
        static unsigned char d[118] = { 71, 73, 70, 56, 55, 97, 16, 0, 16, 0, 179, 0, 0, 0, 0, 0, 0, 206, 156, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 44, 0, 0, 0, 0, 16, 0, 16, 0, 0, 4, 43, 80, 200, 73, 171, 189, 19, 96, 171, 55, 237, 25, 248, 125, 64,
32, 74, 98, 105, 114, 85, 224, 158, 167, 96, 194, 23, 64, 123, 25, 158, 235, 66, 108, 179, 161, 210, 45, 243, 170, 192, 84, 162, 8, 0, 59, };
        BigGIFHelper(theClient, d, 118);
    }
    else if (stricmp(File, "e.gif") == 0) {
        static unsigned char e[124] = { 71, 73, 70, 56, 55, 97, 16, 0, 16, 0, 179, 0, 0, 0, 0, 0, 0, 206, 156, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 44, 0, 0, 0, 0, 16, 0, 16, 0, 0, 4, 49, 16, 136, 57, 65, 184, 146, 138, 92, 111, 224, 21, 103, 97, 218,
102, 118, 95, 153, 137, 0, 120, 190, 101, 165, 185, 179, 26, 203, 118, 76, 195, 184, 158, 223, 160, 29, 69, 164, 35, 242, 90, 41, 153, 107, 20,
92, 122, 64, 17, 0, 59, };
        BigGIFHelper(theClient, e, 124);
    }
    else if (stricmp(File, "f.gif") == 0) {
        static unsigned char f[125] = { 71, 73, 70, 56, 55, 97, 16, 0, 16, 0, 179, 0, 0, 0, 0, 0, 0, 206, 156, 181, 181, 181, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 44, 0, 0, 0, 0, 16, 0, 16, 0, 0, 4, 50, 16, 200, 73, 171, 20, 56, 107, 125, 183, 199, 221, 183, 133,
224, 52, 2, 92, 160, 6, 104, 22, 2, 235, 218, 10, 111, 172, 206, 181, 141, 227, 54, 235, 206, 180, 24, 144, 68, 51, 113, 128, 162, 82, 242,
180, 60, 90, 158, 146, 8, 0, 59, };
        BigGIFHelper(theClient, f, 125);
    }
    else if (stricmp(File, "g.gif") == 0) {
        static unsigned char g[121] = { 71, 73, 70, 56, 55, 97, 16, 0, 16, 0, 179, 0, 0, 0, 0, 0, 0, 206, 156, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 44, 0, 0, 0, 0, 16, 0, 16, 0, 0, 4, 46, 16, 136, 57, 65, 184, 146, 138, 92, 111, 224, 21, 103, 97, 218,
102, 118, 95, 153, 137, 0, 120, 190, 101, 165, 185, 179, 26, 203, 118, 76, 195, 184, 158, 223, 160, 29, 69, 116, 171, 241, 128, 71, 160, 48,
22, 1, 0, 59, };
        BigGIFHelper(theClient, g, 121);
    }
    else if (stricmp(File, "qw.gif") == 0) {
        static unsigned char qw[38] = { 71, 73, 70, 56, 55, 97, 6, 0, 6, 0, 128, 0, 0, 0, 0, 0, 255, 255, 255, 44, 0, 0, 0, 0, 6, 0, 6, 0, 0,
2, 5, 140, 143, 169, 203, 5, 0, 59, };
        BigGIFHelper(theClient, qw, 38);
    }
    else if (stricmp(File, "qa.gif") == 0) {
        static unsigned char qa[42] = { 71, 73, 70, 56, 55, 97, 6, 0, 6, 0, 128, 0, 0, 0, 0, 0, 255, 255, 255, 44, 0, 0, 0, 0, 6, 0, 6, 0, 0,
2, 9, 140, 13, 23, 121, 140, 207, 128, 84, 161, 0, 59, };
        BigGIFHelper(theClient, qa, 42);
    }
    else if (stricmp(File, "qb.gif") == 0) {
        static unsigned char qb[43] = { 71, 73, 70, 56, 55, 97, 6, 0, 6, 0, 128, 0, 0, 0, 0, 0, 255, 255, 255, 44, 0, 0, 0, 0, 6, 0, 6, 0, 0,
2, 10, 132, 31, 25, 176, 204, 11, 225, 67, 42, 20, 0, 59, };
        BigGIFHelper(theClient, qb, 43);
    }
    else if (stricmp(File, "qc.gif") == 0) {
        static unsigned char qc[44] = { 71, 73, 70, 56, 55, 97, 6, 0, 6, 0, 128, 0, 0, 0, 0, 0, 255, 255, 255, 44, 0, 0, 0, 0, 6, 0, 6, 0, 0,
2, 11, 132, 31, 25, 176, 204, 11, 213, 115, 201, 132, 2, 0, 59, };
        BigGIFHelper(theClient, qc, 44);
    }
    else if (stricmp(File, "qd.gif") == 0) {
        static unsigned char qd[43] = { 71, 73, 70, 56, 55, 97, 6, 0, 6, 0, 128, 0, 0, 0, 0, 0, 255, 255, 255, 44, 0, 0, 0, 0, 6, 0, 6, 0, 0,
2, 10, 140, 13, 23, 9, 140, 171, 18, 148, 45, 20, 0, 59, };
```

```
        BigGIFHelper(theClient, qd, 43);
    }
    else if (stricmp(File, "qe.gif") == 0) {
        static unsigned char qe[42] = { 71, 73, 70, 56, 55, 97, 6, 0, 6, 0, 128, 0, 0, 0, 0, 0, 255, 255, 255, 44, 0, 0, 0, 0, 6, 0, 6, 0, 0,
2, 9, 68, 96, 120, 107, 203, 1, 213, 59, 166, 0, 59, };
        BigGIFHelper(theClient, qe, 42);
    }
    else if (stricmp(File, "qf.gif") == 0) {
        static unsigned char qf[42] = { 71, 73, 70, 56, 55, 97, 6, 0, 6, 0, 128, 0, 0, 0, 0, 0, 255, 255, 255, 44, 0, 0, 0, 0, 6, 0, 6, 0, 0,
2, 9, 132, 15, 161, 154, 198, 225, 224, 122, 179, 0, 59, };
        BigGIFHelper(theClient, qf, 42);
    }
    else if (stricmp(File, "qg.gif") == 0) {
        static unsigned char qg[42] = { 71, 73, 70, 56, 55, 97, 6, 0, 6, 0, 128, 0, 0, 0, 0, 0, 255, 255, 255, 44, 0, 0, 0, 0, 6, 0, 6, 0, 0,
2, 9, 68, 96, 120, 107, 203, 1, 21, 116, 179, 0, 59, };
        BigGIFHelper(theClient, qg, 42);
    }
    else if (stricmp(File, "xw.gif") == 0) {
        static unsigned char xw[38] = { 71, 73, 70, 56, 55, 97, 6, 0, 6, 0, 128, 0, 0, 0, 0, 0, 255, 255, 255, 44, 0, 0, 0, 0, 6, 0, 6, 0, 0,
2, 5, 140, 143, 169, 203, 5, 0, 59, };
        BigGIFHelper(theClient, xw, 38);
    }
    else if (stricmp(File, "xa.gif") == 0) {
        static unsigned char xa[91] = { 71, 73, 70, 56, 55, 97, 6, 0, 6, 0, 179, 0, 0, 0, 0, 0, 132, 255, 132, 255, 165, 148, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 44, 0, 0, 0, 0, 6, 0, 6, 0, 0, 4, 16, 112, 72, 9, 230, 0, 98, 226, 124, 129, 191, 65, 128, 1, 225, 22,
1, 0, 59, };
        BigGIFHelper(theClient, xa, 91);
    }
    else if (stricmp(File, "xb.gif") == 0) {
        static unsigned char xb[88] = { 71, 73, 70, 56, 55, 97, 6, 0, 6, 0, 179, 0, 0, 0, 0, 0, 132, 255, 132, 255, 165, 148, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 44, 0, 0, 0, 0, 6, 0, 6, 0, 0, 4, 13, 16, 200, 41, 64, 8, 160, 222, 92, 167, 212, 24, 200, 69, 0, 59,
};
        BigGIFHelper(theClient, xb, 88);
    }
    else if (stricmp(File, "xc.gif") == 0) {
        static unsigned char xc[90] = { 71, 73, 70, 56, 55, 97, 6, 0, 6, 0, 179, 0, 0, 0, 0, 0, 0, 173, 0, 132, 255, 132, 156, 24, 0, 255, 165,
148, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 44, 0, 0, 0, 0, 6, 0, 6, 0, 0, 4, 15, 16, 200, 73, 128, 16, 243, 130, 48, 55, 175, 219, 52, 72, 99, 4, 0, 59, };
        BigGIFHelper(theClient, xc, 90);
    }
    else if (stricmp(File, "xd.gif") == 0) {
        static unsigned char xd[90] = { 71, 73, 70, 56, 55, 97, 6, 0, 6, 0, 179, 0, 0, 0, 0, 0, 0, 173, 0, 156, 24, 0, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 44, 0, 0, 0, 0, 6, 0, 6, 0, 0, 4, 15, 48, 4, 32, 4, 152, 20, 104, 123, 169, 149, 89, 167, 97, 86, 4, 0, 59, };
        BigGIFHelper(theClient, xd, 90);
    }
    else if (stricmp(File, "xe.gif") == 0) {
        static unsigned char xe[89] = { 71, 73, 70, 56, 55, 97, 6, 0, 6, 0, 179, 0, 0, 0, 0, 0, 0, 173, 0, 156, 24, 0, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 44, 0, 0, 0, 0, 6, 0, 6, 0, 0, 4, 14, 80, 8, 32, 41, 176, 55, 100, 16, 116, 231, 219, 53, 89, 17, 0, 59, };
        BigGIFHelper(theClient, xe, 89);
    }
    else if (stricmp(File, "xf.gif") == 0) {
        static unsigned char xf[89] = { 71, 73, 70, 56, 55, 97, 6, 0, 6, 0, 179, 0, 0, 0, 0, 0, 0, 214, 0, 255, 123, 107, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 44, 0, 0, 0, 0, 6, 0, 6, 0, 0, 4, 14, 16, 200, 25, 2, 16, 2, 212, 155, 231, 196, 154, 5, 110, 17, 0, 59, };
        BigGIFHelper(theClient, xf, 89);
    }
    else if (stricmp(File, "xg.gif") == 0) {
        static unsigned char xg[91] = { 71, 73, 70, 56, 55, 97, 6, 0, 6, 0, 179, 0, 0, 0, 0, 0, 0, 173, 0, 156, 24, 0, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 44, 0, 0, 0, 0, 6, 0, 6, 0, 0, 4, 16, 80, 8, 32, 41, 176, 3, 132, 59, 70, 216, 157, 7, 132, 25, 57, 68, 0, 59, };
        BigGIFHelper(theClient, xg, 91);
    }
    else if (stricmp(File, "mon.gif") == 0) {
        static unsigned char mon[2675] = { 71, 73, 70, 56, 57, 97, 67, 0, 220, 0, 247, 0, 0, 74, 74, 74, 82, 173, 90, 82, 173, 99, 82, 173,
107, 255, 255, 198, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255,
                255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255,
                255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255,
                255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 33, 249, 4, 1, 0, 0, 4, 0, 44, 0, 0, 0, 0, 67, 0, 220, 0, 64,
8, 254, 0, 9, 8, 28, 72, 176, 160, 193, 131, 8, 19, 42, 92, 200, 176, 225, 192, 1, 16, 5, 64, 28, 224, 176, 162, 69, 132, 19, 37, 78, 28, 160,
49, 226, 70, 136, 23, 67, 10, 204, 248, 177, 35, 201, 146, 31, 57, 166, 148, 40, 210, 160, 71, 149, 47, 59, 202, 140, 121, 18, 230, 204, 136,
0, 46, 218, 60, 121, 179, 39, 207, 159, 52, 113, 2, 200, 185, 240, 229, 206, 160, 3, 134, 2, 224, 57, 20, 232, 81, 152, 16, 149, 38, 84, 169,
81, 98, 213, 157, 2, 148, 54, 165, 202, 81, 235, 82, 174, 86, 61, 94, 77, 170, 245, 32, 87, 176, 98, 85, 122, 77, 155, 181, 44, 65, 175, 112,
227, 18, 45, 136, 54, 44, 90, 181, 91, 195, 182, 149, 218, 82, 97, 93, 172, 128, 175, 94, 229, 219, 87, 161, 213, 195, 28, 245, 10, 64, 188,
248, 48, 99, 198, 137, 11, 27, 92, 156, 56, 177, 227, 202, 141, 45, 99, 126, 204, 153, 162, 100, 129, 153, 67, 99, 222, 108, 153, 178, 233,
211, 165, 45, 15, 253, 44, 112, 104, 227, 215, 176, 99, 203, 158, 29, 155, 48, 235, 214, 174, 105, 235, 222, 189, 216, 246, 109, 2, 178, 189,
242, 166, 237, 149, 161,
                128, 0, 199, 147, 35, 15, 112, 80, 121, 242, 226, 160, 151, 59, 151, 78, 29, 122, 242, 235, 210, 177, 47, 94, 94, 124, 187,
114, 232, 192, 169, 254, 123, 119, 254, 220, 109, 248, 241, 217, 145, 147, 71, 46, 117, 189, 245, 244, 218, 179, 195, 45, 24, 95, 251, 122,
244, 206, 115, 223, 159, 238, 157, 253, 124, 131, 226, 5, 200, 159, 128, 4, 14, 56, 93, 92, 12, 21, 168, 160, 129, 11, 114, 135, 96, 72, 0, 48,
40, 97, 131, 114, 173, 246, 155, 92, 199, 45, 231, 95, 133, 114, 253, 230, 144, 134, 25, 130, 231, 225, 103, 1, 138, 56, 226, 133, 255, 157,
40, 89, 77, 42, 142, 88, 147, 137, 45, 186, 100, 212, 140, 52, 66, 69, 146, 139, 52, 154, 84, 163, 142, 54, 210, 88, 24, 82, 99, 61, 21, 228,
144, 70, 85, 101, 33, 67, 71, 249, 4, 100, 145, 76, 66, 21, 164, 111, 116, 141, 165, 100, 87, 90, 49, 165, 148, 83, 68, 146, 117, 100, 148,
129, 165, 69, 229, 149, 96, 21, 7, 24, 135, 48, 18, 240, 151, 96, 48, 173, 21, 166, 121, 3, 117, 104, 209, 153, 108, 117, 73, 85, 153, 172,
```

```
133, 118, 217, 105, 157, 161, 246, 26, 71, 45, 226, 185, 217, 157, 126, 234, 9, 153, 85, 191, 41, 38, 40, 160, 136, 106, 6, 40, 159, 183, 41,
42, 168, 163, 129, 46, 154, 25, 148, 125, 9, 55, 220, 165,
        181, 109, 233, 97, 112, 152, 234, 70, 103, 95, 193, 149, 21, 97, 167, 189, 185, 85, 223, 169, 210, 1, 168, 29, 120, 252, 213,
167, 30, 171, 254, 234, 197, 138, 159, 120, 101, 185, 103, 94, 171, 178, 202, 42, 162, 125, 240, 237, 151, 155, 124, 90, 245, 231, 42, 122, 41,
14, 212, 224, 132, 200, 30, 27, 192, 131, 7, 41, 155, 236, 179, 210, 185, 249, 33, 180, 212, 210, 42, 109, 75, 74, 85, 203, 31, 153, 115, 181,
152, 109, 0, 32, 114, 75, 105, 140, 5, 105, 232, 224, 184, 228, 54, 4, 34, 184, 159, 166, 91, 148, 81, 238, 126, 198, 98, 188, 63, 26, 213, 46,
189, 102, 217, 203, 38, 190, 35, 237, 184, 18, 74, 32, 165, 91, 147, 191, 0, 111, 164, 163, 135, 3, 247, 168, 48, 143, 12, 79, 36, 111, 146,
53, 41, 41, 101, 196, 25, 117, 251, 38, 196, 75, 58, 217, 228, 148, 154, 34, 137, 177, 144, 27, 99, 217, 100, 199, 24, 105, 108, 242, 148, 89,
162, 28, 21, 186, 253, 170, 236, 100, 94, 30, 125, 229, 242, 76, 251, 210, 117, 242, 79, 106, 218, 4, 230, 205, 72, 213, 220, 178, 200, 105,
86, 25, 147, 208, 32, 155, 236, 179, 153, 118, 37, 125, 214, 94, 48, 91, 37, 38, 154, 92, 113, 104, 150, 210, 104, 26, 41, 180, 94, 79, 83, 36,
110, 177, 15, 233, 21, 103, 212, 87, 207, 89, 179, 184,
        72, 122, 125, 23, 214, 96, 162, 13, 37, 203, 101, 47, 237, 246, 88, 75, 121, 125, 111, 75, 103, 191, 93, 183, 70, 115, 183, 20,
169, 254, 163, 124, 219, 105, 154, 197, 155, 66, 218, 247, 161, 127, 90, 54, 34, 162, 138, 13, 46, 184, 223, 140, 126, 198, 56, 225, 144, 7,
234, 168, 227, 169, 61, 62, 104, 228, 164, 53, 38, 217, 229, 134, 86, 158, 103, 162, 18, 145, 92, 145, 105, 131, 35, 94, 250, 159, 161, 139,
110, 209, 168, 164, 182, 222, 24, 219, 14, 5, 235, 58, 169, 176, 87, 100, 233, 236, 158, 214, 30, 18, 238, 188, 29, 93, 88, 168, 172, 207, 158,
183, 69, 192, 103, 235, 250, 240, 4, 189, 214, 92, 166, 124, 29, 191, 239, 126, 2, 54, 103, 45, 97, 184, 226, 250, 30, 126, 213, 31, 71, 64,
173, 211, 31, 153, 61, 124, 215, 163, 42, 108, 121, 171, 217, 74, 189, 120, 226, 195, 58, 235, 250, 199, 213, 122, 160, 169, 240, 229, 154, 31,
155, 188, 174, 15, 236, 168, 247, 151, 31, 255, 248, 175, 238, 42, 43, 244, 245, 51, 31, 254, 234, 183, 63, 174, 57, 75, 65, 7, 156, 78, 117,
184, 70, 128, 4, 58, 240, 89, 215, 42, 151, 182, 30, 72, 160, 8, 170, 138, 130, 19, 188, 142, 5, 17, 130, 65, 10, 86, 40, 36, 29, 76, 150, 212,
10, 19, 194, 18, 141, 144, 53, 194, 89, 151, 10, 51,
        212, 62, 178, 169, 40, 46, 44, 60, 215, 214, 248, 53, 144, 12, 69, 139, 76, 52, 108, 86, 12, 151, 229, 187, 28, 38, 15, 92,
254, 44, 68, 30, 191, 128, 72, 68, 33, 250, 112, 123, 12, 60, 162, 66, 152, 165, 196, 38, 34, 204, 96, 158, 113, 226, 187, 110, 36, 197, 41,
122, 164, 138, 86, 84, 9, 22, 253, 2, 69, 35, 210, 11, 37, 94, 140, 81, 194, 82, 2, 56, 26, 22, 44, 97, 38, 17, 0, 191, 206, 72, 48, 40, 6, 76,
140, 110, 140, 227, 24, 25, 166, 34, 54, 46, 236, 95, 5, 195, 209, 28, 241, 40, 199, 26, 53, 170, 143, 119, 180, 99, 195, 86, 52, 147, 66, 6,
101, 98, 128, 193, 74, 189, 102, 214, 164, 143, 77, 172, 140, 13, 113, 100, 200, 50, 38, 49, 178, 132, 100, 146, 69, 99, 100, 207, 32, 137, 17,
68, 102, 18, 104, 46, 211, 157, 153, 120, 246, 73, 74, 138, 172, 118, 166, 76, 101, 41, 159, 130, 202, 85, 166, 12, 148, 64, 73, 29, 39, 127,
230, 202, 168, 96, 242, 149, 47, 233, 161, 42, 173, 246, 21, 168, 192, 108, 151, 186, 172, 165, 150, 122, 217, 145, 157, 225, 18, 108, 170,
219, 229, 48, 137, 57, 17, 99, 222, 82, 40, 162, 59, 164, 83, 114, 86, 76, 103, 166, 242, 104, 112, 242, 154, 44, 155, 182, 178, 43, 81, 237,
76, 101, 186, 155, 219, 212, 100, 23,
        114, 202, 201, 105, 73, 132, 19, 90, 224, 226, 165, 172, 153, 141, 105, 27, 20, 39, 154, 204, 137, 183, 178, 156, 101, 107,
148, 254, 82, 167, 217, 232, 217, 77, 77, 225, 179, 102, 242, 108, 103, 216, 234, 233, 155, 127, 114, 210, 110, 103, 179, 231, 61, 199, 118,
194, 162, 124, 179, 106, 253, 20, 232, 218, 194, 136, 208, 119, 118, 83, 162, 170, 251, 209, 67, 227, 84, 79, 140, 206, 178, 47, 21, 181, 219,
55, 195, 216, 54, 136, 138, 52, 78, 36, 117, 104, 34, 45, 10, 209, 109, 126, 20, 164, 27, 13, 232, 89, 44, 169, 162, 207, 157, 110, 111, 132,
58, 145, 164, 48, 231, 55, 67, 169, 241, 112, 160, 139, 212, 99, 74, 247, 211, 58, 61, 106, 79, 145, 251, 220, 97, 140, 170, 40, 207, 237, 244,
166, 202, 219, 28, 85, 120, 186, 56, 199, 88, 213, 112, 82, 197, 105, 80, 21, 103, 167, 200, 252, 78, 169, 133, 235, 170, 88, 17, 181, 185,
203, 17, 181, 170, 87, 61, 77, 89, 181, 106, 86, 177, 58, 85, 53, 191, 115, 170, 228, 68, 163, 213, 169, 82, 230, 165, 126, 57, 170, 229, 236,
26, 214, 65, 101, 180, 33, 72, 213, 107, 226, 198, 42, 169, 191, 50, 36, 120, 188, 195, 148, 40, 19, 130, 216, 196, 246, 206, 176, 135, 109,
172, 99, 103, 179, 88, 198, 26, 111, 178, 196, 129, 108, 67, 110, 135, 217, 215, 244,
        240, 34, 178, 235, 236, 235, 42, 187, 89, 209, 194, 230, 179, 34, 49, 173, 103, 73, 107, 156, 76, 57, 54, 165, 8, 41, 254, 30,
239, 96, 187, 188, 211, 134, 182, 117, 180, 157, 12, 243, 46, 75, 59, 212, 94, 164, 120, 68, 113, 158, 40, 101, 83, 219, 213, 54, 15, 183, 71,
251, 222, 98, 116, 136, 29, 245, 5, 80, 87, 240, 27, 22, 175, 164, 55, 30, 231, 254, 239, 186, 240, 140, 142, 116, 177, 59, 153, 255, 89, 151,
127, 253, 185, 94, 175, 228, 215, 191, 246, 120, 247, 86, 216, 197, 110, 118, 207, 51, 222, 250, 117, 231, 188, 231, 99, 31, 124, 189, 135,
190, 250, 86, 183, 61, 171, 66, 175, 124, 195, 171, 223, 246, 98, 151, 123, 247, 141, 175, 248, 52, 216, 95, 2, 98, 207, 188, 1, 182, 144, 248,
174, 91, 172, 234, 181, 119, 47, 192, 201, 47, 130, 239, 243, 95, 255, 1, 48, 189, 198, 131, 110, 134, 177, 71, 94, 120, 118, 75, 185, 233,
237, 77, 124, 100, 23, 98, 247, 90, 216, 191, 251, 133, 175, 253, 228, 203, 68, 246, 174, 88, 185, 215, 161, 48, 135, 19, 12, 165, 12, 150,
112, 91, 73, 180, 177, 142, 5, 212, 226, 26, 222, 56, 132, 61, 246, 241, 142, 119, 188, 65, 33, 255, 88, 91, 69, 54, 242, 144, 15, 200, 195,
36, 210, 231, 200, 7, 252, 32, 96, 161, 140, 44, 41, 59, 100, 201,
        16, 76, 114, 66, 176, 188, 32, 43, 131, 150, 203, 3, 242, 178, 72, 34, 68, 229, 38, 7, 25, 66, 100, 94, 50, 87, 14, 81, 68,
230, 21, 18, 209, 205, 220, 58, 17, 12, 137, 168, 64, 15, 55, 116, 68, 115, 86, 142, 153, 103, 232, 174, 20, 198, 240, 159, 57, 116, 13, 117,
246, 44, 102, 26, 14, 218, 206, 154, 141, 23, 157, 55, 148, 232, 33, 190, 143, 181, 238, 90, 244, 122, 183, 40, 16, 21, 230, 86, 69, 235, 154,
52, 165, 15, 125, 233, 19, 189, 185, 133, 144, 230, 215, 153, 41, 61, 234, 45, 150, 58, 36, 1, 1, 0, 59, };
    BigGIFHelper(theClient, mon, 2675);
}
else if (stricmp(File, "pwr.gif") == 0) {
    static unsigned char pwr[2124] = { 71, 73, 70, 56, 57, 97, 56, 0, 148, 0, 131, 0, 0, 0, 0, 0, 0, 49, 99, 255, 57, 8, 255, 66, 16, 255,
74, 16, 255, 82, 24, 255, 90, 24, 255, 99, 24, 255, 107, 33, 255, 115, 33, 255, 123, 41, 255, 132, 41, 255, 140, 41, 255, 148, 49, 255, 255,
255, 255, 255, 255, 33, 255, 11, 78, 69, 84, 83, 67, 65, 80, 69, 50, 46, 48, 3, 1, 232, 3, 0, 33, 249, 4, 0, 3, 0, 0, 0, 44, 0, 0, 0, 0, 0, 56, 0,
148, 0, 3, 4, 251, 48, 200, 73, 171, 189, 56, 235, 205, 187, 255, 96, 40, 142, 100, 105, 158, 104, 170, 174, 108, 235, 190, 112, 44, 87, 192,
60, 2, 181, 13, 2, 77, 174, 115, 184, 70, 239, 7, 20, 26, 125, 68, 11, 207, 40, 68, 38, 39, 75, 230, 240, 73, 137, 74, 157, 73, 235, 149, 42,
209, 110, 159, 65, 233, 151, 232, 21, 79, 127, 101, 243, 121, 150, 86, 99, 95, 109, 183, 45, 46, 151, 209, 235, 240, 59, 190, 165, 87, 175, 87,
125, 126, 111, 39, 129, 126, 127, 132, 134, 137, 99, 41, 133, 134, 131, 37, 141, 123, 140, 138, 148, 143, 144, 148, 137, 150, 36, 75, 12, 13,
157, 159, 158, 161, 160, 163, 135, 147, 152, 102, 12, 154, 155, 161, 172, 157, 173, 66, 174, 174, 77, 112, 76, 178, 176, 82, 178, 174, 170, 55,
162, 183, 177, 189, 175, 187,
        188, 160, 192, 191, 196, 169, 48, 0, 198, 190, 175, 189, 194, 55, 203, 13, 11, 204, 174, 211, 200, 201, 172, 214, 183, 210,
183, 211, 158, 207, 188, 222, 157, 222, 228, 66, 218, 165, 128, 227, 220, 12, 12, 211, 214, 237, 161, 11, 157, 224, 34, 202, 241, 204, 229,
218, 244, 33, 246, 218, 238, 220, 218, 198, 93, 139, 209, 15, 160, 193, 127, 238, 246, 241, 131, 135, 240, 32, 55, 28, 58, 120, 184, 115, 199,
16, 30, 68, 52, 242, 38, 58, 84, 72, 168, 251, 221, 196, 5, 25, 65, 114, 52, 1, 96, 34, 67, 138, 11, 70, 66, 2, 201, 178, 37, 75, 149, 155, 50,
74, 211, 40, 146, 75, 73, 151, 46, 97, 198, 196, 153, 211, 38, 207, 150, 58, 111, 252, 124, 201, 37, 64, 73, 5, 11, 144, 42, 77, 154, 146, 202,
81, 144, 72, 153, 50, 13, 186, 67, 106, 212, 171, 53, 209, 60, 93, 202, 117, 106, 68, 166, 93, 195, 54, 101, 163, 160, 172, 82, 179, 103, 205,
38, 165, 154, 1, 0, 215, 178, 96, 225, 166, 101, 123, 193, 45, 218, 187, 120, 239, 210, 85, 146, 183, 175, 94, 59, 102, 19, 160, 21, 28, 120,
176, 130, 189, 124, 13, 227, 37, 140, 22, 113, 21, 5, 130, 35, 67, 158, 44, 185, 178, 227, 199, 146, 41, 107, 214, 124, 25, 74, 130, 207, 144,
63, 71, 6, 45, 90, 116, 103, 40, 161, 41, 87, 214,
        252, 249, 180, 209, 210, 165, 17, 192, 150, 109, 122, 142, 96, 217, 10, 112, 235, 190, 125, 219, 245, 107, 217, 180, 131, 127,
22, 158, 0, 129, 111, 0, 195, 69, 19, 39, 126, 60, 121, 113, 229, 208, 159, 251, 54, 138, 160, 122, 241, 234, 192, 173, 99, 175, 126, 60, 187,
247, 235, 192, 175, 79, 7, 176, 189, 188, 121, 238, 104, 206, 171, 55, 158, 126, 253, 249, 233, 212, 221, 151, 135, 79, 94, 62, 118, 248, 93,
170, 27, 208, 207, 251, 31, 129, 129, 139, 100, 248, 39, 224, 128, 251, 225, 71, 3, 127, 251, 237, 199, 94, 81, 0, 36, 40, 96, 130, 6, 42, 241,
224, 131, 17, 210, 96, 192, 133, 254, 97, 88, 225, 129, 24, 94, 88, 96, 81, 93, 36, 40, 226, 127, 32, 134, 232, 161, 1, 7, 144, 200, 96, 131,
39, 30, 112, 64, 1, 27, 26, 117, 162, 135, 5, 188, 184, 33, 139, 5, 208, 120, 97, 141, 17, 226, 144, 227, 143, 6, 0, 89, 64, 142, 248, 1, 32,
100, 144, 72, 10, 105, 160, 145, 65, 14, 217, 228, 143, 78, 46, 57, 36, 148, 84, 54, 217, 227, 148, 88, 102, 9, 35, 24, 90, 118, 89, 161, 145,
4, 20, 16, 230, 152, 98, 150, 121, 101, 152, 83, 162, 89, 230, 150, 96, 16, 224, 230, 152, 110, 138, 233, 230, 151, 101, 146, 105, 231, 141,
112, 202, 169, 39, 158, 111, 246, 249, 102, 140,
        0, 248, 41, 40, 159, 130, 254, 201, 96, 161, 134, 218, 132, 232, 156, 32, 6, 74, 192, 0, 143, 70, 58, 64, 140, 70, 73, 106, 41,
165, 149, 90, 10, 41, 166, 0, 12, 224, 233, 167, 159, 114, 10, 42, 168, 162, 142, 234, 41, 166, 70, 153, 58, 105, 137, 169, 14, 32, 128, 171,
167, 178, 218, 170, 167, 175, 162, 218, 234, 171, 161, 202, 218, 41, 172, 181, 234, 202, 171, 171, 182, 166, 250, 106, 175, 190, 14, 61, 187,
170, 172, 194, 10, 32, 64, 176, 201, 18, 91, 172, 178, 204, 2, 160, 44, 180, 200, 118, 225, 234, 178, 213, 26, 53, 45, 179, 93, 80, 155, 173,
180, 222, 86, 11, 46, 183, 218, 98, 155, 173, 182, 228, 162, 123, 110, 23, 233, 170, 187, 46, 128, 223, 182, 107, 212, 186, 20, 68, 0, 0, 33,
249, 4, 0, 100, 0, 0, 0, 44, 0, 0, 0, 0, 56, 0, 148, 0, 3, 4, 251, 48, 200, 73, 171, 189, 56, 235, 205, 187, 255, 96, 40, 142, 100, 105, 158,
104, 170, 174, 108, 235, 190, 112, 44, 207, 116, 109, 223, 42, 128, 123, 128, 190, 111, 128, 70, 195, 247, 187, 4, 133, 196, 34, 229, 136, 84,
90, 152, 67, 231, 82, 72, 141, 74, 161, 212, 228, 15, 155, 117, 114, 187, 197, 111, 85, 91, 19, 143, 119, 230, 51, 46, 173, 46, 87, 223, 112,
50, 140, 253, 150, 187, 232, 117, 26, 62,
        46, 219, 243, 231, 112, 129, 130, 118, 40, 126, 129, 132, 39, 134, 135, 119, 130, 141, 139, 44, 138, 131, 144, 142, 148, 121,
147, 13, 12, 152, 154, 153, 156, 155, 158, 157, 136, 34, 145, 129, 12, 161, 162, 154, 168, 153, 169, 66, 170, 173, 77, 45, 80, 170, 84, 178,
172, 112, 153, 166, 162, 157, 158, 159, 181, 157, 165, 140, 186, 193, 189, 181, 154, 184, 167, 194, 174, 187, 173, 198, 199, 171, 170, 11, 195,
13, 208, 152, 204, 32, 0, 174, 211, 153, 208, 207, 210, 66, 211, 213, 214, 220, 218, 181, 217, 168, 208, 224, 225, 221, 219, 234, 236, 11, 160,
49, 215, 158, 235, 190, 222, 210, 183, 240, 227, 208, 250, 237, 221, 218, 232, 233, 251, 2, 242, 91, 240, 47, 4, 0, 129, 8, 213, 21, 52, 200,
64, 95, 67, 123, 14, 247, 45, 100, 152, 48, 32, 193, 53, 13, 23, 184, 211, 184, 81, 251, 227, 175, 53, 17, 165, 233, 211, 55, 145, 196, 65,
142, 40, 83, 150, 28, 113, 109, 164, 72, 145, 31, 65, 166, 156, 121, 17, 13, 77, 149, 97, 110, 162, 92, 201, 82, 167, 70, 158, 162, 22, 40, 16,
74, 116, 168, 81, 160, 6, 137, 42, 29, 42, 20, 233, 135, 131, 70, 139, 74, 117, 202, 1, 170, 212, 168, 71, 111, 0, 80, 192, 213, 40, 87, 161,
95, 185, 82, 205, 176, 181, 104, 216, 179, 89, 245, 116, 93, 203, 150,
        237, 216, 39, 109, 227, 182, 125, 59, 97, 43, 215, 4, 114, 227, 38, 160, 27, 192, 110, 87, 188, 127, 215, 2, 94, 203, 119, 43,
222, 195, 10, 16, 43, 78, 204, 184, 48, 99, 192, 136, 31, 223, 149, 92, 56, 193, 97, 203, 151, 45, 39, 198, 172, 121, 111, 159, 197, 138, 65,
227, 229, 219, 151, 115, 2, 4, 166, 79, 167, 182, 76, 26, 0, 94, 212, 10, 96, 195, 214, 44, 155, 245, 12, 215, 168, 85, 231, 70, 141, 96, 183,
101, 212, 164, 75, 231, 86, 77, 156, 248, 238, 224, 174, 141, 99, 246, 253, 251, 55, 242, 222, 188, 161, 159, 134, 30, 29, 184, 158, 233, 213,
117, 247, 198, 142, 224, 57, 245, 239, 224, 161, 7, 239, 27, 190, 60, 245, 241, 0, 204, 155, 71, 175, 62, 252, 120, 9, 233, 123, 27, 144, 79,
31, 193, 252, 238, 107, 228, 27, 216, 207, 251, 191, 255, 251, 37, 246, 245, 39, 224, 126, 255, 1, 184, 159, 125, 8, 206, 103, 64, 129, 21, 0,
48, 32, 127, 12, 62, 241, 224, 130, 87, 76, 72, 161, 23, 19, 70, 40, 225, 128, 26, 54, 152, 161, 20, 125, 61, 216, 97, 93, 6, 20, 192, 161, 23,
5, 152, 152, 162, 138, 38, 118, 8, 0, 139, 44, 150, 88, 128, 134, 47, 174, 104, 163, 141, 17, 214, 40, 227, 142, 55, 22, 88, 163, 141, 4, 164,
24, 36, 142, 107, 220, 104, 228, 138,
        12, 254, 120, 36, 145, 107, 16, 224, 228, 147, 80, 66, 153, 227, 144, 84, 22, 16, 228, 149, 52, 58, 105, 37, 1, 3, 112, 249,
228, 0, 89, 126, 41, 228, 0, 66, 18, 144, 37, 153, 95, 166, 233, 98, 148, 91, 98, 25, 70, 148, 112, 154, 249, 102, 156, 79, 142, 8, 0, 157, 78,
218, 233, 100, 151, 124, 122, 41, 231, 155, 3, 240, 25, 40, 151, 96, 122, 65, 168, 151, 124, 142, 8, 95, 160, 140, 14, 90, 168, 18, 0, 52, 42,
169, 162, 125, 73, 202, 40, 165, 149, 90, 250, 40, 164, 154, 110, 202, 169, 165, 152, 86, 42, 192, 0, 163, 6, 26, 106, 164, 129, 150, 122, 42,
169, 163, 10, 32, 64, 168, 153, 182, 234, 169, 23, 174, 214, 10, 107, 164, 174, 146, 10, 171, 168, 182, 130, 88, 87, 173, 175, 250, 90, 23,
171, 187, 246, 38, 213, 171, 176, 240, 29, 139, 44, 0, 174, 22, 107, 108, 176, 200, 38, 11, 109, 180, 204, 58, 107, 172, 181, 215, 70, 91, 23,
```

```
182, 125, 105, 187, 173, 183, 240, 129, 59, 65, 4, 0, 33, 254, 80, 70, 73, 76, 69, 32, 73, 68, 69, 78, 84, 73, 84, 89, 13, 10, 67, 114, 101,
97, 116, 101, 100, 32, 111, 114, 32, 109, 111, 100, 105, 102, 105, 101, 100, 32, 98, 121, 13, 10, 80, 114, 101, 45, 105, 110, 115, 116, 97,
108, 108, 101, 100, 32, 85, 115, 101,
                114, 13, 10, 80, 114, 101, 45, 105, 110, 115, 116, 97, 108, 108, 101, 100, 32, 67, 111, 109, 112, 97, 110, 121, 0, 33, 254,
234, 85, 78, 82, 69, 71, 73, 83, 84, 69, 82, 69, 68, 32, 83, 72, 65, 82, 69, 87, 65, 82, 69, 13, 10, 13, 10, 65, 115, 115, 101, 109, 98, 108,
101, 100, 32, 119, 105, 116, 104, 32, 71, 73, 70, 32, 67, 111, 110, 115, 116, 114, 117, 99, 116, 105, 111, 110, 32, 83, 101, 116, 58, 13, 10,
13, 10, 65, 108, 99, 104, 101, 109, 121, 32, 77, 105, 110, 100, 119, 111, 114, 107, 115, 32, 73, 110, 99, 46, 13, 10, 66, 111, 120, 32, 53, 48,
48, 13, 10, 66, 101, 101, 116, 111, 110, 44, 32, 79, 78, 13, 10, 76, 48, 71, 32, 49, 65, 48, 13, 10, 67, 65, 78, 65, 68, 65, 46, 13, 10, 13,
10, 104, 116, 116, 112, 58, 47, 47, 119, 119, 119, 46, 109, 105, 110, 100, 119, 111, 114, 107, 115, 104, 111, 112, 46, 99, 111, 109, 13, 10,
13, 10, 84, 104, 105, 115, 32, 99, 111, 109, 109, 101, 110, 116, 32, 119, 105, 108, 108, 32, 110, 111, 116, 32, 97, 112, 112, 101, 97, 114, 32,
105, 110, 32, 102, 105, 108, 101, 115, 32, 99, 114, 101, 97, 116, 101, 100, 32, 119, 105, 116, 104, 32, 97, 32, 114, 101, 103, 105, 115, 116,
101, 114, 101, 100, 32, 118, 101,
                114, 115, 105, 111, 110, 46, 0, 33, 255, 11, 71, 73, 70, 67, 79, 78, 110, 98, 49, 46, 48, 2, 2, 0, 14, 9, 0, 2, 0, 3, 0, 0, 0,
0, 0, 0, 0, 0, 0, 9, 112, 119, 114, 49, 46, 103, 105, 102, 0, 14, 9, 0, 2, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 9, 112, 119, 114, 50, 46, 103, 105,
102, 0, 0, 59, };
        BigGIFHelper(theClient, pwr, 2124);
    }
    else if (stricmp(File, "pwr2.gif") == 0) {
        static unsigned char pwr2[2124] = { 71, 73, 70, 56, 57, 97, 56, 0, 148, 0, 131, 0, 0, 0, 0, 0, 0, 49, 99, 255, 57, 8, 255, 66, 16, 255,
74, 16, 255, 82, 24, 255, 90, 24, 255, 99, 24, 255, 107, 33, 255, 115, 33, 255, 123, 41, 255, 132, 41, 255, 140, 41, 255, 148, 49, 255, 255,
255, 255, 255, 255, 33, 255, 11, 78, 69, 84, 83, 67, 65, 80, 69, 50, 46, 48, 3, 1, 232, 3, 0, 33, 249, 4, 0, 3, 0, 0, 0, 44, 0, 0, 0, 0, 56, 0,
148, 0, 3, 4, 251, 48, 200, 73, 171, 189, 56, 235, 205, 187, 255, 96, 40, 142, 100, 105, 158, 104, 170, 174, 108, 235, 190, 112, 44, 87, 192,
60, 2, 181, 13, 2, 77, 174, 115, 184, 70, 239, 7, 20, 26, 125, 68, 11, 207, 40, 68, 38, 39, 75, 230, 240, 73, 137, 74, 157, 73, 235, 149, 42,
209, 110, 159, 65, 233, 151, 232, 21, 79, 127, 101, 243, 121, 150, 86, 99, 95, 109, 183, 45, 46, 151, 209, 235, 240, 59, 190, 165, 87, 175, 87,
125, 126, 111, 39, 129, 126, 127, 132, 134, 137, 99, 41, 133, 134, 131, 37, 141, 123, 140, 138, 148, 143, 144, 148, 137, 150, 36, 75, 12, 13,
157, 159, 158, 161, 160, 163, 135, 147, 152, 102, 12, 154, 155, 161, 172, 157, 173, 66, 174, 174, 77, 112, 76, 178, 176, 82, 178, 174, 170, 55,
162, 183, 177, 189, 175, 187,
                188, 160, 192, 191, 196, 169, 48, 0, 198, 190, 175, 189, 194, 55, 203, 13, 11, 204, 174, 211, 200, 201, 172, 214, 183, 210,
183, 211, 158, 207, 188, 222, 157, 222, 228, 66, 218, 165, 128, 227, 220, 12, 12, 211, 214, 237, 161, 11, 157, 224, 34, 202, 241, 204, 229,
218, 244, 33, 246, 218, 238, 220, 218, 198, 93, 139, 209, 15, 160, 193, 127, 238, 246, 241, 131, 135, 240, 32, 55, 28, 58, 120, 184, 115, 199,
16, 30, 68, 52, 242, 38, 58, 84, 72, 168, 251, 221, 196, 5, 25, 65, 114, 52, 1, 96, 34, 67, 138, 11, 70, 66, 2, 201, 178, 37, 75, 149, 155, 50,
74, 211, 40, 146, 75, 73, 151, 46, 97, 198, 196, 153, 211, 38, 207, 150, 58, 111, 252, 124, 201, 37, 64, 73, 5, 11, 144, 42, 77, 154, 146, 202,
81, 144, 72, 153, 50, 13, 186, 67, 106, 212, 171, 53, 209, 60, 93, 202, 117, 106, 68, 166, 93, 195, 54, 101, 163, 160, 172, 82, 179, 103, 205,
38, 165, 154, 1, 0, 215, 178, 96, 225, 166, 101, 123, 193, 45, 218, 187, 120, 239, 210, 85, 146, 183, 175, 94, 59, 102, 19, 160, 21, 28, 120,
176, 130, 189, 124, 13, 227, 37, 140, 22, 113, 21, 5, 130, 35, 67, 158, 44, 185, 178, 227, 199, 146, 41, 107, 214, 124, 25, 74, 130, 207, 144,
63, 71, 6, 45, 90, 116, 103, 40, 161, 41, 87, 214,
                252, 249, 180, 209, 210, 165, 17, 192, 150, 109, 122, 142, 96, 217, 10, 112, 235, 190, 125, 219, 245, 107, 217, 180, 131, 127,
22, 158, 0, 129, 111, 0, 195, 69, 19, 39, 126, 60, 121, 113, 229, 208, 159, 251, 54, 138, 160, 122, 241, 234, 192, 173, 99, 175, 126, 60, 187,
247, 235, 192, 175, 79, 7, 176, 189, 188, 121, 238, 104, 206, 171, 55, 158, 126, 253, 249, 233, 212, 221, 151, 135, 79, 94, 62, 118, 248, 93,
170, 27, 208, 207, 251, 31, 129, 129, 139, 100, 248, 39, 224, 128, 251, 225, 71, 3, 127, 251, 237, 199, 94, 81, 0, 36, 40, 96, 130, 6, 42, 241,
224, 131, 17, 210, 96, 192, 133, 254, 97, 88, 225, 129, 24, 94, 88, 96, 81, 93, 36, 40, 226, 127, 32, 134, 232, 161, 1, 7, 144, 200, 96, 131,
39, 30, 112, 64, 1, 27, 26, 117, 162, 135, 5, 188, 184, 33, 139, 5, 208, 120, 97, 141, 17, 226, 144, 227, 143, 6, 0, 89, 64, 142, 248, 1, 32,
100, 144, 72, 10, 105, 160, 145, 65, 14, 217, 228, 143, 78, 46, 57, 36, 148, 84, 54, 217, 227, 148, 88, 102, 9, 35, 24, 90, 118, 89, 161, 145,
4, 20, 16, 230, 152, 98, 150, 121, 101, 152, 83, 162, 89, 230, 150, 96, 16, 224, 230, 152, 110, 138, 233, 230, 151, 101, 146, 105, 231, 141,
112, 202, 169, 39, 158, 111, 246, 249, 102, 140,
                0, 248, 41, 40, 159, 130, 254, 201, 96, 161, 134, 218, 132, 232, 156, 32, 6, 74, 192, 0, 143, 70, 58, 64, 140, 70, 73, 106, 41,
165, 149, 90, 10, 41, 166, 0, 12, 224, 233, 167, 159, 114, 10, 42, 168, 162, 142, 234, 41, 166, 70, 153, 58, 105, 137, 169, 14, 32, 128, 171,
167, 178, 218, 170, 167, 175, 162, 218, 234, 171, 161, 202, 218, 41, 172, 181, 234, 202, 171, 171, 182, 166, 250, 106, 175, 190, 14, 61, 187,
170, 172, 194, 10, 32, 64, 176, 201, 18, 91, 172, 178, 204, 2, 160, 44, 180, 200, 118, 225, 234, 178, 213, 26, 53, 45, 179, 93, 80, 155, 173,
180, 222, 86, 11, 46, 183, 218, 98, 155, 173, 182, 228, 162, 123, 110, 23, 233, 170, 187, 46, 128, 223, 182, 107, 212, 186, 20, 68, 0, 0, 33,
249, 4, 0, 127, 0, 0, 0, 44, 0, 0, 0, 0, 56, 0, 148, 0, 3, 4, 251, 48, 200, 73, 171, 189, 56, 235, 205, 187, 255, 96, 40, 142, 100, 105, 158,
104, 170, 174, 108, 235, 190, 112, 44, 207, 116, 109, 223, 42, 128, 123, 128, 190, 111, 128, 70, 195, 247, 187, 4, 133, 196, 34, 229, 136, 84,
90, 152, 67, 231, 82, 72, 141, 74, 161, 212, 228, 15, 155, 117, 114, 187, 197, 111, 85, 91, 19, 143, 119, 230, 51, 46, 173, 46, 87, 223, 112,
50, 140, 253, 150, 187, 232, 117, 26, 62,
                46, 219, 243, 231, 112, 129, 130, 118, 40, 126, 129, 132, 39, 134, 135, 119, 130, 141, 139, 44, 138, 131, 144, 142, 148, 121,
147, 13, 12, 152, 154, 153, 156, 155, 158, 157, 136, 34, 145, 129, 12, 161, 162, 154, 168, 153, 169, 66, 170, 173, 77, 45, 80, 170, 84, 178,
172, 112, 153, 166, 162, 157, 158, 159, 181, 157, 165, 140, 186, 193, 189, 181, 154, 184, 167, 194, 174, 187, 173, 198, 199, 171, 170, 11, 195,
13, 208, 152, 204, 32, 0, 174, 211, 153, 208, 207, 210, 66, 211, 213, 214, 220, 218, 181, 217, 168, 208, 224, 225, 221, 219, 234, 236, 11, 160,
49, 215, 158, 235, 190, 222, 210, 183, 240, 227, 208, 250, 237, 221, 218, 232, 233, 251, 2, 242, 91, 240, 47, 4, 0, 129, 8, 213, 21, 52, 200,
64, 95, 67, 123, 14, 247, 45, 100, 152, 48, 32, 193, 53, 13, 23, 184, 211, 184, 81, 251, 227, 175, 53, 17, 165, 233, 211, 55, 145, 196, 65,
142, 40, 83, 150, 28, 113, 109, 164, 72, 145, 31, 65, 166, 156, 121, 17, 13, 77, 149, 97, 110, 162, 92, 201, 82, 167, 70, 158, 162, 22, 40, 16,
74, 116, 168, 81, 160, 6, 137, 42, 29, 42, 20, 233, 135, 131, 70, 139, 74, 117, 202, 1, 170, 212, 168, 71, 111, 0, 80, 192, 213, 40, 87, 161,
95, 185, 82, 205, 176, 181, 104, 216, 179, 89, 245, 116, 93, 203, 150,
                237, 216, 39, 109, 227, 182, 125, 59, 97, 43, 215, 4, 114, 227, 38, 160, 27, 192, 110, 87, 188, 127, 215, 2, 94, 203, 119, 43,
222, 195, 10, 16, 43, 78, 204, 184, 48, 99, 192, 136, 31, 223, 149, 92, 56, 193, 97, 203, 151, 45, 39, 198, 172, 121, 111, 159, 197, 138, 65,
227, 229, 219, 151, 115, 2, 4, 166, 79, 167, 182, 76, 26, 0, 94, 212, 10, 96, 195, 214, 44, 155, 245, 12, 215, 168, 85, 231, 70, 141, 96, 183,
101, 212, 164, 75, 231, 86, 77, 156, 248, 238, 224, 174, 141, 99, 246, 253, 251, 55, 242, 222, 188, 161, 159, 134, 30, 29, 184, 158, 233, 213,
117, 247, 198, 142, 224, 57, 245, 239, 224, 161, 7, 239, 27, 190, 60, 245, 241, 0, 204, 155, 71, 175, 62, 252, 120, 9, 233, 123, 27, 144, 79,
31, 193, 252, 238, 107, 228, 27, 216, 207, 251, 191, 255, 251, 37, 246, 245, 39, 224, 126, 255, 1, 184, 159, 125, 8, 206, 103, 64, 129, 21, 0,
48, 32, 127, 12, 62, 241, 224, 130, 87, 76, 72, 161, 23, 19, 70, 40, 225, 128, 26, 54, 152, 161, 20, 125, 61, 216, 97, 93, 6, 20, 192, 161, 23,
5, 152, 152, 162, 138, 38, 118, 8, 0, 139, 44, 150, 88, 128, 134, 47, 174, 104, 163, 141, 17, 214, 40, 227, 142, 55, 22, 88, 163, 141, 4, 164,
24, 36, 142, 107, 220, 104, 228, 138,
                12, 254, 120, 36, 145, 107, 16, 224, 228, 147, 80, 66, 153, 227, 144, 84, 22, 16, 228, 149, 52, 58, 105, 37, 1, 3, 112, 249,
228, 0, 89, 126, 41, 228, 0, 66, 18, 144, 37, 153, 95, 166, 233, 98, 148, 91, 98, 25, 70, 148, 112, 154, 249, 102, 156, 79, 142, 8, 0, 157, 78,
218, 233, 100, 151, 124, 122, 41, 231, 155, 3, 240, 25, 40, 151, 96, 122, 65, 168, 151, 124, 142, 8, 95, 160, 140, 14, 90, 168, 18, 0, 52, 42,
169, 162, 125, 73, 202, 40, 165, 149, 90, 250, 40, 164, 154, 110, 202, 169, 165, 152, 86, 42, 192, 0, 163, 6, 26, 106, 164, 129, 150, 122, 42,
169, 163, 10, 32, 64, 168, 153, 182, 234, 169, 23, 174, 214, 10, 107, 164, 174, 146, 10, 171, 168, 182, 130, 88, 87, 173, 175, 250, 90, 23,
171, 187, 246, 38, 213, 171, 176, 240, 29, 139, 44, 0, 174, 22, 107, 108, 176, 200, 38, 11, 109, 180, 204, 58, 107, 172, 181, 215, 70, 91, 23,
182, 125, 105, 187, 173, 183, 240, 129, 59, 65, 4, 0, 33, 254, 80, 70, 73, 76, 69, 32, 73, 68, 69, 78, 84, 73, 84, 89, 13, 10, 67, 114, 101,
97, 116, 101, 100, 32, 111, 114, 32, 109, 111, 100, 105, 102, 105, 101, 100, 32, 98, 121, 13, 10, 80, 114, 101, 45, 105, 110, 115, 116, 97,
108, 108, 101, 100, 32, 85, 115, 101,
                114, 13, 10, 80, 114, 101, 45, 105, 110, 115, 116, 97, 108, 108, 101, 100, 32, 67, 111, 109, 112, 97, 110, 121, 0, 33, 254,
234, 85, 78, 82, 69, 71, 73, 83, 84, 69, 82, 69, 68, 32, 83, 72, 65, 82, 69, 87, 65, 82, 69, 13, 10, 13, 10, 65, 115, 115, 101, 109, 98, 108,
101, 100, 32, 119, 105, 116, 104, 32, 71, 73, 70, 32, 67, 111, 110, 115, 116, 114, 117, 99, 116, 105, 111, 110, 32, 83, 101, 116, 58, 13, 10,
13, 10, 65, 108, 99, 104, 101, 109, 121, 32, 77, 105, 110, 100, 119, 111, 114, 107, 115, 32, 73, 110, 99, 46, 13, 10, 66, 111, 120, 32, 53, 48,
48, 13, 10, 66, 101, 101, 116, 111, 110, 44, 32, 79, 78, 13, 10, 76, 48, 71, 32, 49, 65, 48, 13, 10, 67, 65, 78, 65, 68, 65, 46, 13, 10, 13,
10, 104, 116, 116, 112, 58, 47, 47, 119, 119, 119, 46, 109, 105, 110, 100, 119, 111, 114, 107, 115, 104, 111, 112, 46, 99, 111, 109, 13, 10,
13, 10, 84, 104, 105, 115, 32, 99, 111, 109, 109, 101, 110, 116, 32, 119, 105, 108, 108, 32, 110, 111, 116, 32, 97, 112, 112, 101, 97, 114, 32,
105, 110, 32, 102, 105, 108, 101, 115, 32, 99, 114, 101, 97, 116, 101, 100, 32, 119, 105, 116, 104, 32, 97, 32, 114, 101, 103, 105, 115, 116,
101, 114, 101, 100, 32, 118, 101,
                114, 115, 105, 111, 110, 46, 0, 33, 255, 11, 71, 73, 70, 67, 79, 78, 110, 98, 49, 46, 48, 2, 2, 0, 14, 9, 0, 2, 0, 3, 0, 0, 0,
0, 0, 0, 0, 0, 0, 9, 112, 119, 114, 49, 46, 103, 105, 102, 0, 14, 9, 0, 2, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 9, 112, 119, 114, 50, 46, 103, 105,
102, 0, 0, 59, };
        BigGIFHelper(theClient, pwr2, 2124);
    }
    else if (stricmp(File, "time.gif") == 0) {
        static unsigned char time2[7271] = { 71, 73, 70, 56, 57, 97, 148, 1, 206, 1, 247, 0, 0, 0, 49, 99, 255, 0, 0, 255, 66, 0, 255, 255,
132, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255,
                255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
```

255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255,—255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 44, 0, 0, 0, 0, 148, 1, 206, 1, 0, 8, 254, 0, 1, 8, 28, 72,
176, 160, 193, 131, 8, 19, 42, 92, 200, 176, 161, 195, 135, 16, 35, 74, 156, 72, 177, 162, 197, 139, 24, 51, 106, 220, 200, 177, 163, 199, 143,
32, 67, 66, 12, 64, 178, 164, 200, 147, 40, 83, 170, 92, 201, 178, 165, 203, 151, 2, 75, 202, 156, 73, 18, 166, 205, 155, 56, 115, 234, 220,
169, 147, 166, 207, 159, 1, 120, 10, 29, 74, 180, 168, 209, 157, 64, 147, 38, 61, 202, 180, 169, 211, 167, 80, 15, 42, 157, 74, 181, 163, 204,
168, 88, 179, 106, 221, 74, 144, 170, 215, 165, 86, 129, 114, 29, 75, 182, 108, 206, 175, 104, 77, 134, 245, 58, 81, 173, 217, 183, 112, 227,
26, 76, 91, 149, 35, 93, 154, 18, 103, 202, 221, 203, 87, 235, 93, 159, 30, 255, 130, 29, 137, 183, 175, 225, 195, 66, 5, 7, 93, 171, 120, 106,
222, 159, 136, 35, 75, 86, 121, 151, 113, 227, 180, 143, 197, 78, 222, 204, 25, 227, 87, 203, 151, 5, 183, 85, 218, 185, 180, 233, 133, 117,
53, 134, 94, 93, 152, 112, 234, 211, 176, 57, 59, 190, 200, 186, 182, 230, 136, 104, 99, 235, 62, 60, 123, 180, 237, 223, 131, 93, 179, 221,
77, 220, 44, 105, 220, 192, 147, 191, 126, 72, 183,
184, 243, 172, 199, 29, 42, 159, 254, 217, 119, 239, 231, 216, 153, 222, 150, 78, 189, 251, 112, 228, 209, 179, 254, 139, 31,
186, 189, 161, 247, 243, 223, 193, 3, 30, 207, 158, 39, 100, 225, 232, 203, 83, 167, 88, 190, 189, 253, 150, 239, 153, 199, 215, 107, 126, 190,
250, 171, 247, 5, 136, 95, 107, 220, 157, 231, 217, 116, 255, 213, 36, 224, 130, 40, 173, 167, 95, 114, 33, 117, 87, 32, 128, 12, 86, 248, 17,
129, 253, 173, 246, 146, 127, 10, 57, 104, 225, 135, 25, 97, 200, 80, 104, 61, 33, 216, 33, 127, 32, 166, 104, 17, 138, 19, 138, 6, 146, 0, 48,
198, 40, 227, 137, 16, 34, 228, 161, 138, 56, 182, 152, 96, 101, 25, 201, 232, 227, 143, 63, 38, 84, 163, 141, 44, 230, 104, 36, 141, 139, 61,
232, 226, 68, 64, 54, 233, 100, 144, 66, 218, 22, 37, 133, 71, 86, 57, 151, 130, 240, 241, 216, 210, 147, 48, 78, 73, 34, 145, 84, 90, 41, 38,
0, 88, 102, 137, 89, 78, 78, 122, 217, 152, 154, 99, 182, 89, 166, 146, 90, 242, 212, 164, 154, 113, 22, 84, 100, 155, 70, 38, 105, 102, 110,
76, 65, 41, 149, 98, 96, 186, 133, 103, 142, 122, 194, 217, 92, 84, 62, 210, 153, 158, 157, 97, 14, 234, 232, 149, 127, 145, 21, 163, 162, 215,
117, 213, 232, 163, 152, 2, 10, 87, 151, 129, 86, 58,
208, 157, 152, 58, 170, 169, 108, 139, 126, 42, 104, 168, 162, 46, 217, 89, 169, 49, 157, 138, 170, 155, 254, 170, 150, 198,
42, 153, 174, 190, 90, 229, 168, 167, 205, 122, 169, 173, 132, 198, 106, 90, 117, 140, 190, 201, 107, 175, 117, 198, 166, 171, 176, 195, 166,
232, 171, 177, 172, 214, 154, 236, 135, 145, 62, 7, 172, 165, 133, 62, 11, 98, 177, 196, 77, 107, 106, 181, 214, 86, 120, 104, 118, 218, 182,
202, 109, 183, 11, 158, 41, 94, 184, 180, 142, 75, 110, 128, 232, 98, 135, 46, 178, 235, 178, 187, 220, 120, 239, 170, 27, 111, 123, 243, 210,
251, 238, 189, 22, 230, 171, 111, 190, 240, 242, 251, 111, 112, 246, 241, 105, 169, 192, 2, 206, 58, 112, 120, 159, 34, 44, 175, 167, 248, 134,
107, 175, 195, 197, 249, 203, 158, 193, 20, 151, 11, 113, 132, 251, 229, 71, 95, 187, 25, 47, 236, 113, 131, 29, 223, 152, 153, 197, 33, 131,
187, 49, 72, 37, 155, 188, 35, 193, 41, 139, 236, 50, 199, 45, 239, 186, 231, 200, 49, 159, 187, 242, 133, 53, 59, 251, 114, 125, 57, 75, 187,
115, 96, 61, 7, 124, 51, 206, 65, 59, 135, 50, 209, 69, 171, 134, 113, 210, 74, 51, 124, 82, 209, 62, 255, 140, 52, 212, 186, 13, 13, 90, 203,
78, 63, 141, 117, 214, 82, 139, 68, 181, 209, 86, 207, 252, 245, 175, 97, 211, 220,
116, 215, 32, 159, 221, 23, 183, 90, 219, 69, 245, 70, 216, 186, 109, 24, 220, 105, 179, 60, 55, 254, 219, 10, 219, 189, 23,
178, 113, 211, 189, 247, 129, 94, 251, 125, 55, 222, 48, 139, 61, 248, 138, 223, 26, 142, 24, 224, 121, 71, 92, 184, 227, 127, 11, 27, 184,
187, 230, 82, 206, 155, 229, 145, 203, 220, 185, 230, 111, 185, 122, 121, 197, 141, 131, 254, 54, 231, 137, 171, 13, 92, 74, 165, 155, 94, 57,
226, 169, 27, 58, 246, 228, 157, 46, 237, 122, 89, 181, 210, 142, 218, 236, 209, 102, 216, 238, 196, 183, 67, 149, 123, 230, 45, 242, 222, 118,
176, 245, 6, 143, 251, 240, 117, 215, 110, 252, 231, 233, 18, 223, 170, 242, 203, 67, 142, 43, 165, 207, 95, 29, 125, 221, 100, 83, 223, 212,
165, 26, 34, 153, 61, 195, 225, 67, 234, 61, 87, 160, 110, 191, 230, 159, 227, 247, 198, 26, 152, 231, 163, 111, 179, 148, 144, 182, 127, 219,
251, 245, 199, 191, 85, 250, 219, 94, 198, 190, 253, 249, 41, 31, 242, 244, 7, 29, 254, 245, 239, 122, 226, 2, 32, 129, 190, 148, 63, 2, 98,
197, 108, 7, 236, 29, 181, 20, 200, 34, 255, 253, 15, 120, 14, 76, 140, 246, 18, 40, 193, 9, 82, 176, 76, 22, 68, 94, 247, 50, 232, 30, 160,
121, 176, 117, 28, 252, 224, 9, 165, 199, 65, 18, 10, 239, 115, 203, 82,
31, 5, 69, 56, 57, 17, 185, 176, 40, 125, 147, 97, 179, 62, 24, 166, 14, 70, 16, 131, 254, 55, 196, 201, 241, 116, 152, 54, 30,
162, 142, 133, 27, 12, 226, 89, 116, 71, 67, 19, 26, 177, 90, 216, 74, 162, 18, 133, 200, 66, 241, 205, 236, 137, 245, 59, 150, 13, 167, 88,
194, 42, 90, 209, 89, 88, 108, 226, 22, 87, 56, 66, 46, 222, 4, 133, 58, 114, 94, 251, 178, 104, 29, 3, 154, 177, 139, 57, 28, 145, 186, 140,
184, 181, 170, 189, 81, 131, 113, 172, 8, 15, 153, 6, 193, 59, 146, 103, 136, 109, 180, 95, 29, 203, 232, 71, 56, 66, 239, 100, 130, 20, 156,
9, 11, 169, 29, 219, 33, 114, 141, 124, 235, 35, 35, 141, 146, 71, 207, 217, 113, 146, 141, 60, 36, 230, 98, 135, 201, 167, 140, 46, 91, 149,
236, 36, 30, 23, 153, 197, 49, 30, 238, 147, 162, 60, 35, 39, 197, 136, 191, 253, 57, 50, 149, 170, 148, 226, 15, 121, 135, 20, 84, 194, 146,
50, 178, 76, 225, 19, 109, 185, 59, 77, 222, 146, 117, 146, 156, 229, 46, 17, 40, 59, 55, 254, 18, 151, 198, 36, 227, 48, 67, 248, 200, 100,
30, 83, 117, 64, 212, 229, 50, 107, 163, 199, 85, 62, 147, 103, 151, 84, 230, 52, 233, 23, 72, 155, 253, 209, 129, 222, 44, 229, 54, 57, 212,
204, 108, 194, 36, 156, 166, 51, 167, 48,
199, 105, 162, 110, 18, 18, 153, 239, 116, 92, 60, 217, 217, 179, 106, 6, 115, 106, 166, 254, 188, 93, 52, 165, 73, 207, 248,
48, 46, 159, 208, 140, 39, 9, 251, 73, 75, 123, 58, 51, 68, 164, 228, 34, 65, 179, 247, 79, 116, 210, 198, 154, 115, 1, 231, 66, 199, 215, 80,
117, 22, 243, 160, 219, 138, 223, 68, 21, 88, 209, 125, 126, 241, 158, 39, 164, 222, 70, 247, 104, 80, 143, 178, 18, 164, 179, 212, 231, 72,
195, 248, 49, 139, 242, 19, 165, 235, 76, 231, 74, 119, 249, 80, 129, 134, 82, 155, 38, 165, 216, 76, 183, 89, 83, 143, 250, 114, 157, 14, 13,
217, 78, 233, 89, 211, 178, 149, 13, 163, 252, 26, 234, 66, 139, 90, 60, 140, 162, 49, 99, 74, 29, 233, 67, 47, 90, 70, 31, 190, 138, 144, 81,
157, 233, 74, 58, 23, 195, 76, 85, 53, 171, 67, 133, 167, 41, 173, 234, 213, 175, 130, 53, 170, 192, 188, 26, 89, 83, 101, 214, 179, 102, 53,
173, 99, 244, 98, 78, 239, 115, 80, 183, 218, 117, 174, 55, 195, 222, 22, 131, 154, 48, 190, 218, 111, 0, 128, 13, 172, 96, 7, 75, 216, 194,
14, 224, 174, 0, 237, 232, 252, 242, 134, 84, 145, 81, 181, 104, 134, 141, 172, 100, 39, 75, 217, 200, 142, 51, 160, 1, 43, 34, 76, 55, 169,
78, 0, 86, 246, 179, 160, 13, 173, 96, 47, 139,
205, 184, 170, 53, 161, 58, 171, 235, 95, 69, 203, 90, 208, 6, 224, 179, 164, 221, 254, 90, 47, 193, 200, 75, 180, 37, 246,
165, 84, 107, 173, 110, 43, 251, 90, 215, 242, 84, 145, 14, 5, 223, 77, 115, 149, 75, 34, 142, 109, 183, 200, 157, 108, 111, 67, 251, 91, 132,
90, 116, 92, 94, 220, 13, 106, 129, 154, 219, 228, 90, 183, 176, 203, 101, 110, 115, 75, 74, 184, 232, 218, 22, 164, 10, 188, 174, 120, 7, 75,
146, 221, 198, 214, 58, 221, 101, 226, 170, 166, 75, 93, 200, 142, 247, 189, 135, 205, 110, 107, 207, 171, 164, 158, 170, 119, 51, 190, 12,
175, 104, 105, 2, 95, 194, 150, 215, 186, 244, 149, 35, 94, 113, 91, 159, 1, 203, 143, 189, 4, 174, 238, 126, 103, 210, 95, 242, 202, 23, 185,
1, 38, 82, 36, 165, 198, 87, 227, 64, 180, 189, 238, 213, 45, 131, 27, 28, 216, 255, 138, 151, 157, 35, 162, 143, 56, 11, 92, 225, 3, 35, 216,
184, 188, 75, 238, 134, 57, 28, 223, 254, 70, 184, 167, 35, 190, 83, 113, 253, 50, 58, 10, 94, 119, 197, 28, 46, 73, 131, 95, 28, 200, 24, 195,
238, 182, 81, 185, 169, 141, 199, 139, 99, 248, 202, 132, 197, 60, 150, 221, 5, 69, 199, 213, 32, 191, 50, 193, 10, 254, 240, 145, 115, 172,
99, 22, 199, 119, 154, 45, 53, 153, 13, 115, 104, 96, 146,
61, 25, 197, 179, 163, 178, 135, 93, 92, 101, 43, 131, 248, 104, 71, 92, 50, 254, 39, 5, 42, 214, 11, 99, 56, 195, 100, 30,
179, 145, 203, 108, 229, 43, 99, 217, 119, 48, 251, 241, 210, 216, 140, 217, 25, 131, 249, 184, 102, 166, 243, 156, 229, 92, 231, 51, 171, 17,
181, 107, 77, 87, 137, 134, 203, 195, 58, 3, 86, 208, 239, 157, 178, 163, 71, 187, 93, 156, 106, 175, 171, 166, 138, 37, 163, 63, 40, 89, 66,
155, 215, 211, 82, 6, 181, 138, 31, 236, 224, 101, 250, 56, 117, 137, 102, 212, 134, 188, 123, 210, 231, 89, 22, 210, 159, 38, 245, 141, 37,
29, 105, 88, 119, 216, 212, 167, 118, 162, 215, 236, 229, 82, 197, 126, 178, 209, 134, 229, 47, 145, 101, 13, 96, 90, 135, 218, 214, 183, 166,
105, 174, 145, 246, 52, 111, 246, 186, 156, 63, 53, 226, 171, 133, 61, 107, 49, 139, 58, 214, 215, 166, 52, 75, 45, 205, 108, 79, 185, 241,
217, 120, 6, 36, 183, 195, 172, 92, 106, 87, 59, 206, 217, 102, 173, 185, 41, 171, 236, 113, 107, 57, 113, 16, 44, 177, 128, 89, 221, 106, 87,
179, 123, 221, 163, 222, 177, 177, 139, 141, 236, 78, 147, 180, 222, 221, 222, 50, 41, 229, 125, 232, 159, 254, 153, 220, 176, 45, 114, 190,
209, 77, 108, 13, 247, 187, 220, 42, 92, 182, 203, 4, 30, 187, 198, 170, 217,
224, 210, 86, 183, 194, 33, 172, 239, 135, 47, 56, 221, 247, 158, 161, 196, 254, 69, 36, 220, 157, 17, 124, 228, 38, 125, 34,
182, 27, 174, 238, 142, 131, 220, 183, 47, 15, 57, 0, 47, 238, 237, 130, 143, 21, 220, 238, 222, 167, 202, 33, 188, 239, 73, 107, 183, 231,
177, 30, 52, 36, 81, 142, 206, 230, 89, 220, 221, 143, 205, 222, 172, 99, 238, 115, 255, 110, 220, 225, 44, 95, 249, 243, 108, 62, 240, 167,
182, 176, 164, 23, 222, 249, 177, 155, 62, 223, 167, 119, 61, 234, 82, 55, 30, 205, 25, 75, 239, 237, 101, 89, 107, 187, 20, 58, 215, 53, 238,
245, 159, 59, 58, 145, 0, 127, 247, 125, 35, 232, 206, 13, 166, 189, 227, 107, 255, 56, 208, 23, 220, 116, 138, 142, 61, 56, 86, 175, 29, 180,
77, 171, 117, 125, 231, 253, 231, 251, 6, 59, 121, 243, 206, 208, 191, 203, 157, 151, 48, 253, 93, 225, 145, 172, 248, 195, 251, 196, 191, 124,
183, 188, 216, 29, 95, 114, 55, 95, 28, 205, 187, 154, 60, 229, 15, 207, 219, 203, 47, 158, 185, 72, 150, 249, 216, 168, 206, 107, 207, 159,
20, 244, 153, 229, 244, 189, 63, 76, 122, 213, 67, 186, 242, 111, 79, 119, 65, 255, 62, 239, 147, 255, 57, 233, 240, 2, 182, 191, 153, 94, 123,
158, 227, 155, 248, 214, 214, 253, 234, 15, 93, 188, 179, 31, 82, 139, 171,
29, 254, 222, 139, 63, 236, 227, 227, 158, 242, 109, 79, 118, 61, 63, 254, 63, 33, 231, 59, 18, 250, 74, 79, 248, 244, 169,
191, 116, 133, 35, 159, 225, 76, 95, 92, 72, 71, 226, 253, 33, 90, 44, 250, 226, 247, 56, 249, 195, 254, 224, 243, 11, 29, 249, 107, 99, 95,
155, 197, 253, 123, 206, 151, 12, 234, 246, 55, 127, 182, 71, 104, 1, 120, 108, 1, 184, 125, 82, 1, 87, 195, 213, 94, 172, 215, 49, 198, 87,
128, 2, 136, 93, 63, 225, 116, 212, 151, 125, 165, 87, 50, 231, 180, 128, 18, 215, 128, 232, 81, 126, 17, 88, 107, 166, 71, 105, 21, 40, 127,
241, 231, 79, 44, 81, 91, 172, 167, 87, 14, 88, 125, 31, 104, 128, 137, 119, 125, 106, 119, 127, 222, 177, 85, 126, 118, 84, 115, 4, 104, 115,
214, 130, 229, 103, 126, 16, 104, 124, 201, 55, 131, 248, 228, 123, 131, 71, 91, 81, 150, 131, 58, 248, 128, 60, 8, 131, 161, 86, 104, 38, 136,
77, 25, 88, 113, 112, 150, 99, 68, 54, 127, 64, 225, 116, 74, 120, 110, 185, 103, 32, 124, 177, 50, 81, 152, 122, 55, 70, 133, 19, 88, 106,
154, 183, 118, 90, 248, 58, 139, 84, 51, 71, 200, 91, 35, 24, 134, 14, 102, 121, 164, 7, 132, 114, 193, 133, 255, 151, 134, 236, 182, 134, 108,
120, 107, 116, 248, 131, 171, 19, 135, 80, 184, 31, 121,
168, 134, 197, 167, 20, 20, 248, 135, 232, 247, 27, 102, 104, 54, 254, 126, 72, 136, 179, 87, 123, 130, 216, 134, 138, 8, 130,
237, 20, 58, 125, 120, 30, 143, 88, 135, 140, 152, 20, 86, 88, 137, 213, 23, 137, 100, 17, 57, 241, 161, 137, 229, 118, 137, 85, 40, 134, 160,
200, 111, 228, 52, 22, 100, 215, 129, 202, 245, 136, 61, 72, 127, 183, 215, 138, 228, 119, 125, 112, 216, 137, 147, 216, 29, 50, 167, 136, 87,
104, 138, 119, 248, 104, 176, 24, 136, 233, 55, 139, 38, 134, 136, 170, 56, 109, 36, 232, 139, 154, 135, 137, 164, 88, 138, 188, 88, 140, 203,
120, 138, 52, 102, 119, 148, 40, 125, 227, 23, 139, 185, 136, 132, 33, 152, 108, 202, 216, 140, 204, 136, 30, 33, 70, 73, 85, 231, 29, 23, 104,
129, 151, 120, 140, 163, 168, 109, 217, 184, 139, 3, 184, 135, 168, 225, 141, 151, 22, 141, 37, 216, 139, 100, 198, 120, 141, 104, 142, 160,
56, 143, 225, 232, 140, 127, 66, 20, 181, 56, 29, 95, 7, 143, 181, 38, 143, 200, 168, 109, 213, 8, 144, 215, 8, 115, 18, 210, 75, 127, 20, 111,
224, 184, 114, 58, 200, 140, 59, 88, 144, 118, 86, 137, 232, 136, 120, 7, 73, 35, 163, 100, 64, 238, 232, 131, 45, 232, 143, 233, 248, 130, 28,
233, 114, 189, 200, 141, 243, 86, 75, 194, 104, 139, 195, 182, 145, 3,
169, 119, 229, 168, 125, 132, 56, 141, 108, 7, 140, 129, 178, 254, 104, 24, 185, 144, 217, 120, 129, 125, 55, 21, 86, 152, 146,
76, 136, 125, 101, 200, 38, 154, 54, 147, 212, 81, 147, 48, 119, 147, 246, 216, 140, 66, 249, 144, 248, 72, 67, 54, 241, 141, 252, 120, 148,
165, 71, 148, 1, 201, 146, 78, 105, 141, 73, 169, 148, 171, 166, 144, 77, 57, 149, 16, 231, 115, 84, 145, 147, 90, 73, 149, 156, 40, 120, 46,

1, 141, 65, 249, 149, 195, 199, 149, 93, 41, 144, 102, 9, 128, 61, 169, 40, 3, 114, 115, 89, 185, 150, 193, 246, 145, 57, 25, 149, 17, 41, 151, 47, 217, 150, 220, 183, 127, 173, 167, 28, 120, 121, 150, 76, 152, 150, 82, 249, 151, 6, 169, 151, 123, 233, 101, 225, 84, 150, 132, 153, 137, 129, 137, 147, 244, 184, 152, 133, 9, 147, 52, 167, 128, 100, 19, 151, 144, 57, 152, 122, 184, 139, 14, 185, 152, 24, 216, 123, 66, 216, 84, 31, 197, 26, 151, 41, 129, 58, 41, 141, 118, 185, 153, 156, 217, 49, 158, 249, 153, 171, 25, 154, 161, 49, 154, 164, 217, 152, 69, 137, 154, 132, 217, 153, 179, 5, 100, 61, 230, 154, 151, 1, 155, 94, 201, 147, 179, 73, 151, 53, 105, 155, 186, 169, 55, 137, 153, 28, 188, 25, 155, 190, 121, 154, 192, 121, 142, 194, 169, 130, 124, 84, 53, 150, 121, 156, 119,

89, 136, 43, 121, 101, 210, 217, 97, 115, 217, 156, 54, 254, 39, 91, 42, 168, 24, 215, 89, 106, 165, 89, 151, 191, 25, 158, 18, 41, 106, 92, 179, 154, 184, 121, 155, 19, 227, 151, 223, 137, 153, 144, 232, 152, 216, 120, 156, 79, 87, 51, 225, 182, 89, 98, 217, 157, 127, 209, 158, 224, 9, 146, 69, 217, 98, 188, 137, 111, 218, 183, 31, 245, 153, 158, 247, 201, 129, 105, 161, 159, 106, 73, 157, 232, 184, 156, 31, 56, 145, 218, 40, 160, 232, 73, 160, 34, 164, 158, 171, 129, 160, 9, 42, 131, 118, 233, 159, 144, 153, 161, 1, 218, 132, 20, 106, 159, 7, 244, 161, 187, 105, 161, 238, 137, 148, 213, 169, 161, 181, 201, 161, 15, 234, 161, 248, 201, 103, 102, 167, 155, 249, 73, 162, 37, 58, 106, 95, 49, 136, 127, 169, 162, 224, 9, 161, 17, 106, 159, 223, 246, 27, 79, 121, 153, 226, 24, 153, 253, 201, 160, 190, 232, 146, 57, 202, 162, 48, 42, 111, 69, 103, 27, 241, 7, 164, 180, 153, 157, 94, 97, 163, 107, 249, 164, 43, 42, 153, 45, 58, 79, 118, 228, 163, 239, 152, 154, 240, 72, 23, 167, 39, 163, 80, 138, 164, 73, 250, 92, 197, 89, 27, 239, 72, 165, 230, 149, 133, 33, 121, 160, 163, 5, 166, 166, 105, 165, 87, 42, 28, 59, 122, 23, 66, 106, 164, 145, 38, 155, 155, 24, 165, 109, 234, 166, 97,

106, 152, 34, 90, 98, 75, 42, 154, 103, 10, 160, 254, 102, 214, 152, 121, 42, 152, 143, 198, 167, 125, 10, 167, 248, 169, 30, 127, 74, 167, 84, 201, 117, 203, 25, 164, 16, 135, 22, 95, 170, 168, 71, 90, 145, 192, 231, 168, 99, 90, 163, 66, 9, 156, 148, 250, 166, 240, 153, 168, 152, 42, 158, 154, 218, 84, 233, 121, 73, 102, 234, 148, 160, 106, 167, 196, 232, 169, 216, 89, 170, 166, 26, 150, 115, 138, 165, 192, 179, 170, 172, 74, 158, 2, 153, 146, 144, 26, 171, 178, 186, 171, 167, 58, 160, 74, 218, 61, 130, 170, 149, 31, 25, 170, 139, 58, 170, 135, 245, 171, 179, 58, 36, 73, 151, 25, 143, 202, 166, 95, 201, 145, 132, 90, 167, 253, 201, 172, 153, 170, 28, 67, 152, 114, 183, 90, 161, 83, 74, 173, 174, 74, 154, 150, 138, 173, 162, 170, 142, 207, 138, 28, 181, 10, 171, 102, 137, 166, 117, 73, 163, 227, 74, 174, 201, 106, 174, 168, 58, 172, 233, 138, 168, 223, 74, 158, 213, 218, 145, 215, 10, 175, 29, 234, 172, 243, 106, 83, 221, 250, 154, 41, 138, 175, 225, 10, 172, 122, 202, 175, 226, 74, 171, 48, 250, 24, 245, 170, 172, 120, 201, 174, 29, 202, 150, 234, 90, 170, 217, 198, 168, 142, 199, 176, 209, 122, 176, 92, 234, 130, 18, 171, 177, 138, 58, 126, 22, 155, 107, 24, 219, 169, 56, 122, 175, 58,

153, 175, 229, 186, 175, 22, 10, 145, 6, 107, 254, 136, 160, 151, 23, 194, 58, 177, 27, 10, 177, 40, 27, 175, 42, 251, 157, 37, 107, 103, 254, 58, 167, 109, 209, 176, 55, 43, 151, 5, 43, 158, 253, 248, 174, 43, 235, 176, 45, 75, 77, 104, 70, 156, 141, 33, 157, 65, 11, 172, 121, 41, 173, 237, 41, 179, 217, 218, 74, 255, 218, 101, 6, 154, 179, 15, 139, 166, 53, 59, 181, 246, 42, 159, 80, 219, 172, 2, 20, 179, 86, 235, 127, 70, 235, 164, 5, 184, 181, 71, 219, 181, 8, 203, 181, 204, 36, 182, 74, 235, 157, 215, 137, 172, 6, 75, 145, 95, 187, 182, 108, 187, 62, 84, 245, 182, 75, 27, 183, 77, 203, 178, 149, 74, 180, 118, 11, 182, 120, 251, 175, 122, 11, 183, 124, 75, 155, 104, 107, 179, 137, 75, 174, 218, 122, 52, 133, 43, 24, 250, 41, 183, 58, 219, 131, 106, 27, 184, 253, 202, 64, 132, 91, 90, 35, 26, 181, 5, 187, 183, 229, 106, 185, 175, 74, 181, 49, 171, 185, 155, 11, 165, 55, 218, 185, 189, 186, 165, 160, 171, 184, 169, 22, 99, 143, 107, 184, 9, 43, 185, 71, 184, 184, 147, 139, 181, 63, 186, 186, 130, 219, 186, 172, 244, 186, 144, 27, 186, 63, 75, 135, 139, 91, 185, 184, 139, 161, 109, 27, 161, 188, 27, 163, 190, 43, 181, 105, 152, 184, 182, 235, 111, 195, 107, 173, 152,

54, 153, 164, 91, 186, 177, 11, 187, 13, 254, 73, 169, 178, 155, 163, 207, 123, 143, 152, 43, 172, 211, 75, 189, 185, 235, 185, 96, 104, 164, 173, 120, 162, 219, 27, 190, 115, 151, 82, 223, 11, 190, 105, 11, 28, 118, 72, 164, 181, 75, 187, 150, 203, 77, 222, 187, 190, 226, 91, 189, 52, 41, 169, 242, 75, 183, 205, 139, 187, 244, 43, 162, 199, 219, 187, 201, 107, 156, 239, 11, 174, 94, 122, 190, 172, 91, 118, 68, 100, 191, 247, 139, 190, 169, 219, 160, 251, 155, 192, 253, 11, 186, 72, 91, 191, 131, 196, 190, 237, 171, 188, 203, 139, 182, 160, 230, 173, 8, 124, 183, 233, 91, 131, 85, 219, 192, 32, 156, 189, 17, 232, 183, 241, 137, 191, 15, 252, 193, 241, 219, 60, 35, 22, 192, 214, 43, 184, 167, 59, 159, 253, 134, 171, 44, 220, 194, 129, 71, 70, 48, 44, 192, 166, 11, 155, 154, 9, 178, 54, 204, 194, 97, 251, 81, 12, 140, 193, 69, 43, 108, 226, 200, 164, 55, 92, 165, 57, 172, 190, 23, 44, 176, 152, 42, 188, 37, 172, 193, 207, 59, 196, 110, 89, 196, 70, 124, 196, 14, 170, 194, 60, 252, 193, 86, 236, 60, 59, 28, 195, 110, 106, 190, 3, 76, 109, 202, 170, 171, 145, 219, 189, 95, 20, 198, 93, 252, 177, 180, 219, 181, 179, 121, 190, 95, 172, 102, 88, 156, 197, 36, 42, 185, 56, 250, 155, 140,

123, 109, 115, 44, 78, 117, 76, 194, 254, 124, 138, 184, 121, 28, 149, 240, 91, 138, 91, 140, 195, 33, 124, 116, 62, 43, 198, 129, 236, 138, 230, 169, 156, 50, 42, 197, 125, 108, 149, 207, 201, 158, 75, 92, 182, 122, 124, 184, 128, 59, 197, 154, 36, 161, 139, 204, 200, 219, 107, 180, 103, 140, 198, 27, 220, 198, 156, 236, 102, 158, 156, 177, 30, 124, 189, 164, 140, 150, 25, 170, 177, 16, 59, 187, 197, 42, 193, 151, 147, 202, 36, 11, 200, 39, 108, 194, 95, 8, 189, 176, 27, 203, 185, 172, 196, 101, 156, 200, 56, 167, 202, 171, 156, 135, 135, 28, 143, 188, 204, 195, 77, 251, 135, 65, 236, 192, 157, 204, 154, 87, 139, 203, 165, 140, 194, 88, 56, 168, 165, 187, 204, 172, 40, 205, 136, 44, 110, 182, 124, 165, 197, 172, 137, 191, 235, 118, 252, 235, 193, 186, 92, 158, 7, 28, 204, 220, 172, 200, 183, 108, 199, 217, 60, 193, 140, 57, 206, 80, 60, 157, 255, 233, 177, 92, 60, 119, 221, 28, 205, 218, 92, 143, 82, 220, 155, 240, 124, 205, 190, 172, 140, 191, 59, 201, 33, 250, 199, 179, 60, 173, 26, 44, 183, 4, 28, 160, 145, 140, 178, 162, 235, 147, 149, 108, 146, 51, 12, 171, 120, 12, 204, 187, 234, 198, 44, 215, 208, 157, 66, 208, 5, 61, 176, 43, 156, 204, 244, 188, 162, 129, 139, 209, 23, 164, 209, 223,

60, 179, 117, 171, 146, 231, 124, 254, 180, 33, 45, 208, 87, 247, 196, 9, 173, 201, 238, 156, 178, 14, 122, 204, 191, 42, 210, 13, 244, 208, 249, 203, 185, 52, 141, 210, 91, 76, 205, 123, 172, 198, 49, 137, 211, 57, 141, 160, 17, 172, 175, 178, 230, 211, 63, 13, 212, 255, 67, 210, 27, 189, 208, 178, 73, 108, 69, 29, 200, 54, 61, 64, 66, 13, 209, 50, 189, 211, 78, 253, 170, 8, 92, 193, 179, 197, 212, 37, 237, 204, 173, 140, 179, 255, 188, 208, 44, 173, 104, 108, 76, 209, 180, 60, 154, 97, 125, 161, 94, 60, 213, 212, 210, 103, 86, 205, 186, 243, 156, 133, 75, 188, 205, 101, 55, 204, 196, 172, 165, 127, 139, 205, 134, 188, 214, 188, 88, 215, 76, 172, 187, 102, 125, 214, 77, 109, 170, 76, 91, 200, 128, 109, 215, 246, 236, 162, 98, 107, 201, 210, 39, 214, 136, 13, 216, 255, 27, 37, 65, 184, 130, 206, 11, 217, 126, 157, 216, 25, 124, 215, 99, 187, 206, 95, 77, 214, 145, 45, 217, 92, 61, 37, 136, 249, 137, 81, 28, 213, 154, 237, 209, 244, 134, 215, 141, 253, 210, 160, 93, 206, 169, 109, 212, 46, 156, 81, 165, 157, 145, 68, 29, 211, 177, 253, 180, 197, 155, 143, 148, 105, 219, 48, 141, 213, 185, 237, 200, 186, 203, 216, 176, 231, 190, 58, 141, 219, 193, 205, 211, 131, 75, 217, 124, 169, 152, 243,

28, 206, 201, 173, 139, 44, 254, 77, 220, 197, 141, 214, 38, 77, 198, 209, 205, 159, 74, 173, 106, 205, 29, 157, 89, 139, 220, 217, 45, 220, 195, 253, 150, 166, 29, 209, 72, 29, 222, 218, 189, 219, 17, 117, 130, 150, 109, 178, 168, 157, 219, 52, 235, 178, 235, 200, 222, 137, 104, 208, 192, 141, 222, 179, 58, 180, 101, 61, 150, 245, 157, 171, 231, 141, 223, 70, 205, 189, 251, 77, 222, 190, 221, 215, 239, 13, 224, 185, 123, 213, 130, 29, 24, 121, 93, 216, 184, 120, 224, 8, 94, 198, 10, 62, 219, 17, 242, 201, 236, 44, 203, 124, 29, 225, 117, 58, 225, 77, 204, 51, 13, 30, 207, 45, 9, 219, 26, 174, 146, 123, 61, 224, 86, 241, 225, 23, 206, 111, 72, 56, 226, 69, 202, 225, 29, 126, 33, 22, 14, 202, 210, 205, 150, 44, 46, 128, 13, 215, 184, 171, 134, 226, 249, 60, 206, 0, 88, 227, 230, 44, 223, 252, 173, 227, 50, 110, 205, 231, 55, 214, 62, 174, 166, 147, 77, 131, 235, 89, 222, 202, 61, 209, 71, 14, 208, 59, 171, 228, 66, 142, 188, 170, 253, 228, 207, 45, 175, 3, 50, 229, 29, 93, 226, 34, 110, 229, 237, 140, 229, 244, 173, 229, 41, 77, 228, 94, 238, 181, 81, 46, 229, 158, 61, 228, 248, 91, 230, 135, 125, 230, 200, 36, 230, 39, 93, 226, 108, 222, 230, 96, 174, 228, 75, 94, 224, 127,

59, 231, 191, 61, 218, 254, 252, 125, 231, 67, 109, 147, 122, 110, 230, 110, 14, 79, 105, 110, 202, 171, 24, 232, 116, 94, 231, 118, 14, 231, 84, 140, 232, 96, 90, 149, 132, 94, 232, 91, 238, 232, 99, 172, 176, 118, 222, 151, 127, 78, 233, 149, 142, 227, 79, 88, 153, 120, 174, 233, 119, 12, 233, 145, 238, 205, 41, 14, 234, 130, 206, 233, 87, 105, 52, 253, 109, 234, 224, 140, 198, 33, 91, 217, 126, 158, 233, 172, 206, 204, 190, 252, 234, 138, 227, 108, 171, 62, 235, 95, 14, 129, 182, 14, 77, 146, 110, 232, 186, 110, 204, 255, 13, 214, 11, 174, 58, 140, 30, 231, 193, 46, 236, 195, 174, 216, 234, 77, 223, 208, 57, 135, 201, 254, 227, 28, 220, 235, 190, 78, 234, 205, 28, 237, 202, 142, 189, 212, 174, 183, 199, 62, 230, 216, 78, 235, 219, 184, 237, 211, 251, 235, 59, 254, 237, 54, 30, 238, 193, 218, 231, 184, 142, 134, 147, 22, 218, 108, 142, 123, 34, 73, 69, 64, 217, 50, 129, 230, 238, 223, 30, 239, 63, 169, 234, 93, 232, 174, 246, 110, 238, 248, 190, 148, 112, 217, 51, 38, 106, 238, 111, 232, 167, 4, 254, 236, 251, 110, 205, 4, 47, 138, 233, 174, 238, 235, 158, 240, 41, 187, 240, 5, 111, 240, 7, 143, 233, 16, 159, 188, 18, 79, 142, 226, 254, 184, 228, 238, 224, 125, 154, 241, 26, 191, 241, 223,

219, 241, 30, 47, 254, 174, 32, 79, 144, 255, 14, 240, 88, 137, 131, 176, 117, 242, 100, 120, 158, 75, 212, 71, 198, 227, 242, 211, 172, 154, 36, 105, 76, 51, 79, 243, 16, 76, 159, 55, 143, 243, 41, 166, 243, 227, 203, 243, 50, 153, 76, 57, 15, 244, 19, 159, 127, 49, 127, 91, 69, 111, 244, 250, 187, 124, 67, 239, 76, 75, 239, 134, 217, 157, 217, 204, 46, 242, 246, 11, 124, 195, 40, 143, 232, 189, 214, 141, 215, 243, 170, 133, 112, 100, 136, 223, 116, 233, 119, 94, 95, 97, 81, 223, 238, 84, 95, 211, 174, 78, 246, 79, 15, 168, 63, 239, 202, 0, 254, 207, 112, 215, 246, 110, 15, 246, 129, 137, 224, 25, 206, 81, 134, 68, 160, 103, 143, 119, 120, 143, 205, 34, 183, 247, 141, 213, 247, 144, 168, 225, 178, 27, 113, 130, 63, 248, 111, 239, 114, 27, 121, 220, 48, 248, 111, 101, 207, 247, 139, 95, 248, 141, 143, 217, 49, 183, 109, 116, 47, 249, 118, 191, 131, 13, 153, 198, 46, 217, 110, 145, 95, 131, 147, 175, 139, 40, 233, 249, 64, 140, 249, 73, 255, 101, 80, 38, 235, 79, 91, 250, 166, 127, 227, 184, 118, 145, 168, 52, 250, 250, 109, 100, 42, 174, 211, 92, 30, 248, 178, 63, 251, 155, 223, 250, 39, 73, 163, 68, 157, 251, 122, 175, 143, 47, 254, 102, 113, 205, 211, 191, 255, 128, 106, 15, 250, 56,

84, 252, 171, 254, 127, 252, 116, 155, 252, 216, 70, 177, 177, 255, 61, 10, 108, 252, 206, 253, 117, 44, 232, 174, 167, 61, 76, 15, 116, 253, 207, 159, 253, 243, 181, 132, 166, 104, 209, 204, 255, 66, 252, 135, 253, 222, 77, 144, 164, 191, 233, 168, 95, 64, 233, 127, 112, 172, 239, 202, 188, 154, 225, 92, 249, 212, 144, 143, 138, 233, 27, 254, 235, 79, 255, 156, 15, 16, 3, 4, 14, 36, 88, 208, 224, 65, 132, 9, 21, 26, 12, 16, 96, 225, 67, 133, 13, 37, 78, 164, 88, 209, 226, 69, 140, 25, 53, 74, 4, 208, 209, 227, 71, 144, 33, 69, 142, 36, 89, 210, 100, 201, 141, 19, 79, 130, 76, 217, 210, 37, 70, 136, 49, 101, 10, 164, 56, 51, 97, 77, 155, 57, 117, 30, 108, 184, 243, 225, 75, 160, 65, 129, 174, 36, 90, 212, 232, 81, 143, 47, 143, 10, 101, 186, 209, 167, 207, 138, 79, 105, 70, 149, 90, 21, 162, 68, 171, 4, 155, 110, 229, 218, 16, 233, 87, 176, 97, 63, 186, 92, 218, 213, 172, 195, 172, 49, 113, 74, 181, 152, 214, 45, 207, 158, 89, 207, 206, 37, 43, 214, 238, 93, 162, 117, 139, 210, 229, 250, 54, 226, 218, 167, 109, 253, 14, 198, 106, 149, 239, 97, 140, 120, 21, 47, 38, 217, 178, 44, 226, 160, 131, 25, 82, 101, 75, 57, 45, 218, 183, 19, 171, 66, 230, 204, 145, 241,

103, 208, 254, 0, 28, 27, 237, 12, 84, 242, 64, 193, 155, 45, 203, 37, 92, 120, 103, 105, 206, 161, 101, 131, 78, 249, 24, 182, 211, 211, 169, 3, 235, 102, 237, 23, 240, 204, 219, 136, 103, 15, 255, 188, 209, 118, 112, 152, 173, 87, 191, 230, 109, 24, 243, 229, 223, 87, 145, 211, 37, 94, 125, 113, 237, 189, 211, 51, 230, 214, 236, 124, 121, 229, 231, 208, 93, 171, 213, 126, 214, 250, 121, 197, 198, 179, 151, 111, 158, 57, 60, 212, 246, 187, 223, 123, 31, 255, 151, 61, 87, 244, 249, 241, 170, 207, 123, 255, 251, 105, 248, 46, 114, 171, 62, 185, 226, 34, 207, 191, 166, 244, 83, 240, 46, 141, 72, 67, 144, 64, 0, 3, 140, 143, 57, 3, 35, 180, 239, 193, 160, 22, 212, 80, 172, 6, 215, 195, 176, 66, 11, 109, 218, 78, 188, 249, 66, 156, 234, 195, 12, 55, 84, 17, 172, 140, 28, 68, 177, 59, 19, 165, 19, 144, 196, 24, 39, 123, 81, 175, 21, 115, 68, 42, 49, 15, 111, 132, 176, 198, 1, 52, 26, 240, 71, 11, 125, 196, 81, 71, 36, 123, 84, 73, 73, 35, 75, 12, 81, 72, 18, 157, 228, 174, 201, 14, 147, 180, 210, 69, 138, 176, 164, 18, 70, 19, 161, 164, 15, 196, 8, 183, 228, 239, 74, 50, 87, 186, 72, 75, 49, 185, 156, 114, 198, 47, 165, 116, 47, 77, 139, 202, 148, 19, 77, 175,

250, 254, 131, 115, 66, 213, 70, 108, 179, 203, 59, 43, 154, 243, 79, 59, 179, 12, 180, 207, 255, 36, 100, 51, 79, 53, 149, 35, 180, 78, 64, 27, 53, 41, 206, 65, 23, 45, 20, 56, 220, 16, 37, 178, 77, 66, 29, 213, 244, 36, 63, 35, 149, 116, 210, 3, 15, 5, 210, 208, 76, 55, 53, 21, 37, 65, 205, 252, 52, 37, 240, 162, 27, 85, 196, 85, 119, 60, 117, 206, 84, 57, 93, 213, 37, 249, 192, 124, 53, 84, 73, 101, 157, 149, 214, 37, 61, 189, 85, 207, 156, 116, 221, 21, 161, 97, 25, 205, 238, 215, 70, 131, 85, 53, 89, 92, 143, 29, 18, 90, 101, 251, 99, 214, 81, 103, 109, 165, 246, 37, 105, 41, 220, 182, 90, 85, 175, 221, 52, 91, 84, 191, 53, 173, 219, 155, 204, 173, 117, 89, 113, 199, 245, 236, 89, 117, 185, 69, 55, 200, 120, 59, 117, 177, 221, 89, 223, 213, 182, 222, 115, 99, 228, 151, 199, 123, 241, 253, 21, 220, 198, 254, 101, 106, 77, 131, 215, 13, 84, 96, 102, 9, 30, 41, 225, 166, 48, 133, 152, 92, 120, 25, 22, 55, 0, 38, 39, 142, 86, 38, 141, 199, 28, 212, 98, 144, 31, 237, 120, 43, 100, 71, 246, 248, 227, 144, 83, 126, 216, 100, 150, 111, 251, 74, 97, 149, 99, 22, 173, 101, 154, 133, 219, 145, 98, 153, 99, 174, 121, 103, 179, 94, 134, 57, 103, 153,

```
                254, 121, 22, 90, 40, 159, 113, 6, 58, 232, 161, 147, 62, 25, 94, 135, 143, 206, 89, 105, 168, 33, 61, 14, 99, 167, 171, 30,
43, 234, 168, 139, 54, 218, 106, 160, 177, 78, 90, 107, 125, 185, 22, 187, 35, 175, 107, 6, 187, 233, 177, 173, 46, 123, 100, 22, 237, 77, 251,
237, 171, 215, 254, 247, 108, 180, 225, 78, 91, 238, 120, 233, 174, 219, 238, 183, 241, 134, 182, 109, 169, 249, 22, 60, 36, 191, 99, 213, 155,
234, 193, 19, 23, 169, 240, 62, 1, 119, 91, 113, 200, 89, 98, 188, 73, 199, 31, 143, 252, 242, 184, 39, 199, 176, 242, 159, 49, 247, 92, 243,
7, 57, 223, 218, 115, 210, 65, 215, 46, 172, 22, 73, 87, 93, 100, 211, 59, 19, 125, 244, 213, 99, 159, 185, 245, 185, 56, 76, 93, 118, 220, 11,
166, 61, 197, 253, 110, 207, 253, 247, 149, 119, 7, 152, 177, 42, 129, 55, 62, 120, 218, 137, 91, 250, 120, 230, 147, 154, 28, 189, 229, 155,
151, 158, 108, 175, 23, 140, 126, 122, 236, 103, 167, 57, 199, 235, 179, 247, 190, 227, 36, 71, 243, 126, 124, 228, 191, 149, 83, 124, 242,
211, 95, 124, 216, 102, 177, 83, 255, 253, 242, 169, 60, 21, 125, 248, 235, 39, 252, 198, 134, 233, 183, 127, 255, 204, 79, 199, 247, 72, 254,
5, 176, 127, 144, 97, 24, 239, 4, 120, 192, 251, 213, 238, 127, 101,
                91, 65, 96, 3, 117, 55, 148, 5, 154, 199, 129, 19, 92, 95, 241, 242, 103, 51, 10, 102, 48, 129, 136, 187, 86, 112, 52, 248,
193, 2, 178, 7, 132, 35, 156, 223, 230, 72, 120, 194, 50, 25, 9, 133, 43, 92, 81, 154, 88, 248, 194, 243, 244, 10, 134, 51, 188, 14, 251, 104,
120, 195, 169, 37, 11, 135, 59, 124, 96, 222, 120, 200, 67, 240, 253, 16, 134, 219, 19, 34, 9, 135, 86, 196, 15, 66, 13, 137, 75, 100, 98, 19,
157, 248, 68, 40, 70, 17, 36, 1, 1, 0, 59, };
        BigGIFHelper(theClient, time2, 7271);
    }
/*
    else if (stricmp(File, "time.gif") == 0) {
        static unsigned char time[1620] = { 71, 73, 70, 56, 57, 97, 35, 0, 47, 0, 247, 0, 0, 255, 255, 255, 247, 247, 247, 247, 222, 222, 239,
239, 239, 239, 214, 214, 231, 231, 222, 231, 214, 214, 222, 222, 222, 222, 206, 206, 214, 222, 222, 214, 214, 206, 214, 198, 198, 206, 206,
206, 206, 198, 189, 206, 189, 189, 198, 198, 198, 189, 189, 189, 189, 181, 181, 181, 189, 198, 181, 189, 181, 173, 173, 173, 173, 165, 165,
165, 181, 189, 165, 156, 156, 156, 173, 181, 156, 156, 156, 148, 173, 173, 148, 140, 140, 140, 165, 165, 140, 156, 165, 140, 140, 140, 132,
165, 165, 123, 156, 156, 115, 156, 165, 115, 115, 115, 107, 148, 156, 107, 140, 140, 107, 132, 123, 99, 132, 140, 99, 99, 99, 90, 132, 148, 90,
82, 82, 82, 123, 132, 82, 82, 82, 74, 123, 140, 74, 115, 123, 66, 115, 123, 66, 107, 107, 66, 66, 66, 57, 115, 140, 57, 115, 132, 57, 107, 115,
49, 107, 132, 41, 90, 115, 33, 99, 140, 33, 99, 132, 33, 33, 33, 24, 99, 140, 24, 90, 132, 16, 16, 16, 8, 8, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 33, 249, 4, 1, 0, 0, 0, 0, 44, 0, 0, 0, 0, 35, 0, 47, 0, 0, 8, 254, 0, 1, 8, 28, 72,
176, 96, 0, 5, 15, 24, 12, 40, 200, 176, 161, 67, 0, 1, 44, 140, 144, 81, 67, 71, 13, 25, 40, 52, 44, 124, 200, 113, 96, 1, 19, 58, 104, 132,
176, 64, 242, 195, 12, 29, 45, 20, 116, 236, 56, 66, 7, 136, 2, 5, 7, 104, 184, 97, 98, 37, 199, 24, 45, 2, 0, 40, 192, 64, 194, 131, 3, 2, 85,
208, 208, 105, 179, 33, 138, 26, 22, 42, 184, 160, 113, 131, 198, 140, 12, 16, 104, 176, 40, 234, 144, 1, 197, 23, 51, 92, 184, 104, 49, 227,
69, 13, 26, 15, 168, 58, 60, 224, 97, 134, 13, 27, 52, 206, 222, 40, 161, 82, 236, 195, 14, 58, 206, 218, 200, 65, 130, 168, 91, 135, 1, 72,
200, 85, 177, 241, 238, 67, 14, 55, 206, 146, 240, 219, 241, 65, 90, 27, 26, 8, 227, 77, 240, 64, 198, 217, 9, 9, 250, 18, 30, 240, 96, 68, 11,
185, 152, 91, 144, 144, 32, 185, 104, 68, 21, 55, 114, 228, 184, 33, 35, 134, 105, 25, 161, 71, 203, 176, 208, 153, 227, 129, 22, 161, 109,
200, 40, 1, 66, 3, 6, 11, 24, 48, 128, 80, 225, 120, 116, 11, 6, 54, 27, 231, 176, 225, 162, 182,
                5, 159, 12, 22, 60, 120, 32, 193, 130, 6, 16, 151, 115, 208, 152, 208, 17, 66, 104, 26, 36, 56, 64, 88, 128, 128, 128, 247,
239, 4, 16, 254, 44, 128, 160, 129, 132, 140, 209, 20, 30, 62, 168, 145, 99, 6, 8, 11, 15, 12, 120, 55, 64, 191, 190, 124, 239, 15, 44, 124,
56, 95, 3, 66, 67, 171, 57, 200, 240, 193, 3, 221, 217, 71, 223, 6, 5, 214, 23, 158, 5, 28, 156, 71, 131, 127, 5, 153, 32, 29, 8, 18, 200, 103,
32, 1, 11, 236, 224, 128, 0, 23, 34, 96, 1, 8, 52, 228, 192, 130, 93, 0, 88, 103, 131, 9, 22, 36, 104, 159, 0, 27, 244, 176, 1, 135, 29, 90,
128, 130, 13, 58, 112, 64, 16, 10, 1, 106, 240, 0, 1, 6, 26, 32, 128, 3, 59, 244, 176, 67, 4, 48, 218, 135, 33, 6, 231, 177, 176, 209, 3, 129,
109, 214, 35, 125, 4, 60, 112, 66, 15, 39, 56, 192, 99, 143, 4, 72, 64, 194, 13, 58, 164, 7, 0, 7, 57, 184, 160, 221, 149, 61, 10, 112, 65, 15,
23, 20, 121, 33, 4, 28, 180, 144, 3, 8, 2, 225, 136, 66, 138, 100, 26, 104, 102, 15, 25, 168, 217, 163, 4, 35, 136, 8, 81, 12, 55, 132, 32, 65,
157, 118, 158, 153, 230, 133, 87, 102, 25, 2, 90, 9, 28, 144, 214, 7, 131, 62, 105, 225, 148, 41, 32, 32, 192, 165,
                151, 46, 224, 35, 126, 31, 216, 112, 3, 4, 76, 218, 240, 65, 3, 132, 214, 39, 128, 8, 61, 164, 154, 130, 3, 17, 180, 234, 128,
8, 254, 48, 84, 224, 99, 3, 28, 208, 208, 165, 137, 163, 150, 122, 31, 12, 169, 246, 202, 195, 175, 60, 244, 122, 66, 171, 26, 180, 160, 195,
4, 161, 230, 42, 233, 169, 189, 198, 122, 193, 179, 21, 156, 128, 67, 6, 8, 32, 64, 171, 12, 199, 58, 42, 106, 164, 79, 134, 183, 2, 15, 48,
88, 138, 169, 0, 17, 44, 32, 128, 119, 13, 116, 122, 195, 4, 1, 0, 42, 168, 164, 80, 98, 72, 128, 158, 243, 201, 39, 65, 167, 52, 36, 0, 0,
142, 40, 194, 27, 175, 174, 246, 241, 233, 231, 151, 237, 113, 64, 170, 191, 8, 19, 208, 128, 6, 42, 228, 48, 130, 64, 161, 146, 96, 1, 194, 9,
67, 48, 194, 13, 235, 14, 196, 66, 14, 42, 96, 176, 35, 197, 221, 46, 128, 193, 198, 35, 14, 52, 65, 96, 40, 170, 8, 178, 124, 30, 110, 89, 99,
132, 19, 210, 185, 50, 203, 247, 134, 88, 50, 65, 10, 204, 80, 48, 5, 11, 0, 220, 33, 121, 46, 72, 7, 97, 65, 134, 133, 249, 129, 5, 61, 251,
236, 221, 2, 250, 93, 118, 131, 151, 13, 77, 16, 34, 13, 38, 96, 176, 221, 125, 224, 241, 56, 30, 6, 36, 152, 117, 67, 98, 28, 65, 160, 179,
13,
                42, 140, 128, 129, 4, 18, 52, 224, 128, 218, 13, 160, 173, 27, 11, 215, 65, 205, 209, 3, 112, 143, 86, 246, 7, 28, 216, 166, 1,
52, 7, 31, 140, 32, 84, 14, 40, 13, 189, 82, 68, 117, 199, 69, 67, 11, 136, 167, 21, 218, 13, 49, 112, 64, 162, 103, 19, 140, 192, 130, 173,
58, 84, 30, 82, 12, 35, 80, 208, 218, 93, 7, 76, 224, 185, 231, 10, 60, 238, 80, 64, 0, 0, 59, };
        BigGIFHelper(theClient, time, 1620);
    }
*/
            else {
                        send(theClient, "HTTP/1.0 404\r\n", 14, 0);
                        send(theClient, "Content-type: ", 14, 0);
                        send(theClient, "text/html", 9, 0);
                        send(theClient, "\r\n", 2, 0);
                        send(theClient, "\r\n", 2, 0);
                        send(theClient, "404 not found\n", 14, 0);
            }

}

void GIFEncode(FILE *out, char *code) {
            char fname[100];
            sprintf(fname, "%s.gif", code);
            FILE *test = fopen(fname, "rb");
            int len = 0;
            int c;
            while ((c = fgetc(test)) != -1)
                        len++;

            fclose(test);
            test = fopen(fname, "rb");

            fprintf(out, "    else if (stricmp(File, \"%s\") == 0) {\n        static unsigned char %s[%d] = { ", fname, code, len);

            len = 0;
            while ((c = fgetc(test)) != -1) {
                        fprintf(out, "%d, ", c);
                        if (++len % 256 == 0)
                                    fprintf(out, "\n                ");
            }

            fprintf(out, "};\n        BigGIFHelper(theClient, %s, %d);\n    }\n", code, len);


            fclose(test);
}

int main() {

            // helper code to generate GIF writing code
            // create a 1x1 pixel GIF in GIF87a uninterlaced format and convert to 77 integers with the code below
            if (0) {
                        FILE *otest = fopen("images.cpp", "w");
                        GIFEncode(otest, "time2");
```

```
                        fclose (otest);
                }

#if __GNUC__ != 0
                int rc;
#else
    WSADATA wsdata;
    int rc = WSAStartup(MAKEWORD(1,1), &wsdata);
                if (rc) {
        printf("WSAStartup failed: error code = %d\n", rc);
        return 1;
    }
#endif

                int Local_Address_Family = AF_INET;
                int Socket_Type = SOCK_STREAM;
#if __GNUC__ != 0
                int Protocol = 0;
#else
                int Protocol = IPPROTO_TCP;
#endif
    u_int s = socket(Local_Address_Family, Socket_Type, Protocol);

                if (s == (u_int)INVALID_SOCKET) {
                        printf("Socket call failed\n");
                        return 1;
                }

                int yes = 1;
                setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (const char *)&yes, sizeof(int));

#if __GNUC__ != 0
#define SOCKADDR_IN sockaddr_in
#endif
                SOCKADDR_IN addr;
    addr.sin_family = Local_Address_Family;
    addr.sin_addr.s_addr = INADDR_ANY;
#if __GNUC__ != 0
    addr.sin_port = htons(1710);
#else
    addr.sin_port = htons(80);
#endif

                rc = bind(s, (const struct sockaddr *) &addr, sizeof(SOCKADDR_IN));

                if (rc == SOCKET_ERROR) {
                        printf("Error at bind()");
                        return 0;
                }

                rc = listen(s, 10);             // 10 is the number of clients that can be queued
                if (rc == SOCKET_ERROR) {
                        printf("Error at listen()");
                        return 0;
                }

                for (int i = 0; i < 200000; i++) {
                        SOCKET theClient = accept(s, NULL, NULL);
                        if (theClient == INVALID_SOCKET) {
                                printf("Error at accept()");
                                return 0;
                        }

                        int total = 0;
                        char *p, buf[10000];
                        do {
                                rc = recv(theClient, buf, sizeof buf, 0);
                                if (rc < 0)
                                        break;
                                p = (char *)(total == 0 ? malloc(rc + 1) : realloc(p, total + rc + 1));
                                memcpy(p + total, buf, rc);
                                total += rc;
                                p[total] = 0;
                        } while (rc == sizeof buf);

                        // printf("%s", p);

                        if (p[0] == 'G' && p[1] == 'E' && p[2] == 'T' && p[3] == ' ') {
                                char *q = p + 4, *e = q;
                                while (*e != 0 && *e != ' ' && *e != '?')
                                        e++;

                                char *file = (char *)malloc(e - q + 1);
                                strncpy(file, q, e - q);
                                file[e - q] = 0;
                                // printf("file %s\n", file);

                                int argc = 0;
                                char **name = NULL;
                                char **value = NULL;

                                while (*e == '?' || *e == '&') {
                                        q = ++e;
                                        while (*e != 0 && *e != '&' && *e != '=' && *e != ' ' && *e != '\r' && *e != '\n')
                                                e++;
                                        if (e != q) {
                                                name = (char **)(argc++ == 0 ? malloc(sizeof(char *) * argc) : realloc(name,
sizeof(char *) * argc));

                                                name[argc - 1] = (char *)malloc(e - q + 1);
                                                strncpy(name[argc - 1], q, e - q);
                                                name[argc - 1][e - q] = 0;
                                                // printf("%s", name[argc - 1]);
                                                if (*e == '=') {
                                                        q = ++e;
                                                        while (*e != 0 && *e != '&' && *e != ' ' && *e != '\r' && *e != '\n')
                                                                e++;
                                                }
                                                else
                                                        e = q;
                                                value = (char **)(argc == 1 ? malloc(sizeof(char *) * argc) : realloc(value,
sizeof(char *) * argc));

                                                value[argc - 1] = (char *)malloc(e - q + 1);
                                                strncpy(value[argc - 1], q, e - q);
                                                value[argc - 1][e - q] = 0;
```

```
                                        // printf("=%s\n", value[argc - 1]);
                                    }
                                }

                                Query qx(argc, name, value);

                                WebServe(theClient, file, qx);

                                if (argc > 0) {
                                    while (argc > 0) {
                                        free(name[argc - 1]);
                                        free(value[argc-- - 1]);
                                    }
                                    free(name);
                                    free(value);
                                }
                                free(file);
                            }

                            free(p);

#if __GNUC__ != 0
                            close(theClient);
#else
                            closesocket(theClient);
#endif
                        }

#if __GNUC__ != 0
                    close(s);
#else
                    closesocket(s);
                    WSACleanup();
#endif
                    return 0;
}
```