Lecture 1

Introduction

$Id: intro.tex,v 1.7 2004/02/16 21:31:14 drcheng Exp $

This book strives to study that elusive mix of theory and practice so important for understanding modern high performance computing. We try to cover it all, from the engineering aspects of computer science (parallel architectures, vendors, parallel languages, and tools) to the mathematical understanding of numerical algorithms, and also certain aspects from theoretical computer science.

Any understanding of the subject should begin with a quick introduction to the current scene in terms of machines, vendors, and performance trends. This sets a realistic expectation of maximal performance and an idea of the monetary price. Then one must quickly introduce at least one, though possibly two or three software approaches so that there is no waste in time in using a computer. Then one has the luxury of reviewing interesting algorithms, understanding the intricacies of how architecture influences performance and how this has changed historically, and also obtaining detailed knowledge of software techniques.

1.1 The machines

We begin our introduction with a look at machines available for high performance computation. We list four topics worthy of further exploration:

The **top500** list: We encourage readers to explore the data on http://www.top500.org. Important concepts are the types of machines currently on the top 500 and what benchmark is used. See Section 1.8 for a case study on how this might be done.

The "**grass-roots**" machines: We encourage readers to find out what machines that one can buy in the 10k–300k range. These are the sorts of machines that one might expect a small group or team might have available. Such machines can be built as a do-it-yourself project or they can be purchased as pre-built rack-mounted machines from a number of vendors. We created a web-site at MIT `http://beowulf.lcs.mit.edu/hpc.html` to track the machines available at MIT.

Special interesting architectures: At the time of writing, the Japanese Earth Simulator and the Virginia Tech Mac cluster are of special interest. It will not be long before they move into the next category:

Interesting historical architectures and trends: To get an idea of history, consider the popular 1990 Michael Crichton novel *Jurassic Park* and the 1993 movie. The novel has the dinosaur theme park controlled by a Cray vector supercomputer. The 1993 movie shows the CM-5 in the background of the control center and even mentions the Thinking Machines Computer, but you could easily miss it if you do not pay attention. The first decade of architecture is captured by the Crichton novel: vector supercomputers. We recommend the Cray supercomputers as an interesting examples. The second decade is characterized by **MPP**s: massively parallel supercomputers. We recommend the CM2 and CM5 for their historical interest. The third decade, that of the cluster, has seen a trend toward ease of availability, deployment and use. The first cluster dates back to 1994 consisting of 16 commodity computers connected by ethernet. Historically, the first beowulf is worthy of study.

When studying architectures, issues of interconnect, processor type and speed, and other nitty gritty issues arise. Sometimes low level software issues are also worthy of consideration when studying hardware.

1.2 The software

The three software models that we introduce quickly are

MPI—The message passing interface. This is the defacto standard for parallel computing though perhaps it is the lowest common denominator. We believe it was originally meant to be the high performance low-level language that libraries and compilers would reduce to. In fact, because it is portably and universally available it has become very much the language of parallel computing.

OpenMP—This less successful language (really language extension) has become popular for so-called shared memory implementations. Those are implementations where the user need not worry about the location of data.

Star-P—Our homegrown software that we hope will make parallel computing significantly easier. It is based on a server which currently uses a MATLAB front end, and either OCTAVE or MATLAB compute engines as well as library calls on the back end.

We also mention that there are any number of software libraries available for special purposes.

Mosix and **OpenMosix** are two technologies which allow for automatic load balancing between nodes of a Linux cluster. The difference between the two is that OpenMosix is released under the GNU Public License, while Mosix is proprietary software. Mosix and OpenMosix are installed as kernel patches (so it is the somewhat daunting task of patching, recompiling, and installing the patched Linux kernel). Once installed on a cluster, processes are automatically migrated from node to node to achieve load balancing. This allows for an exceptionally simple way to run embarrassingly parallel jobs, by simply backgrounding them with the ampersand (&) in the shell. For example:

```
#! /bin/sh
for i in 1 2 3 4 5 6 7 8 9 10
    do ./monte-carlo \$i &
done
wait
echo "All processes done."
```

Although all ten `monte-carlo` processes (each executing with a different command-line parameter) initially start on the same processor, the Mosix or OpenMosix system will automatically migrate the processes to different nodes of the cluster by capturing the entire state of the running program, sending it over the network, and restarting it from where it left off on a different node. Unfortunately, interprocess communication is difficult. It can be done through the standard Unix methods, for example, with sockets or via the file system.

The Condor Project, developed at the University of Wisconsin at Madison, is a batch queing system with the an interesting feature.

> [Condor can] effectively harness wasted CPU power from otherwise idle desktop workstations. For instance, Condor can be configured to only use desktop machines where

> the keyboard and mouse are idle. Should Condor detect that a machine is no longer available (such as a key press detected), in many circumstances Condor is able to transparently produce a checkpoint and migrate a job to a different machine which would otherwise be idle.[1]

Condor can run parallel computations across multiple Condor nodes using PVM or MPI, but (for now) using MPI requires dedicated nodes that cannot be used as desktop machines.

1.3 The Reality of High Performance Computing

There are probably a number of important issues regarding the reality of parallel computing that all too often is learned the hard way. You may not often find this in previously written textbooks.

Parallel computers may not give a speedup of p but you probably will be happy to be able to solve a larger problem in a reasonable amount of time. In other words if your computation can already be done on a single processor in a reasonable amount of time, you probably cannot do much better.

If you are deplying a machine, worry first about heat, power, space, and noise, not speed and performance.

1.4 Modern Algorithms

This book covers some of our favorite modern algorithms. One goal of high performance computing is very well defined, that is, to find faster solutions to larger and more complex problems. This area is a highly interdisciplinary area ranging from numerical analysis and algorithm design to programming languages, computer architectures, software, and real applications. The participants include engineers, computer scientists, applied mathematicians, and physicists, and many others.

We will concentrate on well-developed and more research oriented parallel algorithms in scientific and engineering that have "traditionally" relied on high performance computing, particularly parallel methods for differential equations, linear algebra, and simulations.

1.5 Compilers

[move to a software section?]

Most parallel languages are defined by adding parallel extensions to well-established sequential languages, such as C and Fortran. Such extensions allow user to specify various levels of parallelism in an application program, to define and manipulate parallel data structures, and to specify message passing among parallel computing units.

Compilation has become more important for parallel systems. The purpose of a compiler is not just to transform a parallel program in a high-level description into machine-level code. The role of compiler optimization is more important. We will always have discussions about: *Are we developing methods and algorithms for a parallel machine or are we designing parallel machines for algorithm and applications*? Compilers are meant to bridge the gap between algorithm design and machine architectures. Extension of compiler techniques to run time libraries will further reduce users' concern in parallel programming.

Software libraries are important tools in the use of computers. The purpose of libraries is to enhance the productivity by providing preprogrammed functions and procedures. Software

[1] http://www.cs.wisc.edu/condor/description.html

libraries provide even higher level support to programmers than high-level languages. Parallel scientific libraries embody expert knowledge of computer architectures, compilers, operating systems, numerical analysis, parallel data structures, and algorithms. They systematically choose a set of basic functions and parallel data structures, and provide highly optimized routines for these functions that carefully consider the issues of data allocation, data motion, load balancing, and numerical stability. Hence scientists and engineers can spend more time and be more focused on developing efficient computational methods for their problems. Another goal of scientific libraries is to make parallel programs portable from one parallel machine platform to another. Because of the lack, until very recently, of non-proprietary parallel programming standards, the development of portable parallel libraries has lagged far behind the need for them. There is good evidence now, however, that scientific libraries will be made more powerful in the future and will include more functions for applications to provide a better interface to real applications.

Due to the generality of scientific libraries, their functions may be more complex than needed for a particular application. Hence, they are less efficient than the best codes that take advantage of the special structure of the problem. So a programmer needs to learn how to use scientific libraries. A pragmatic suggestion is to use functions available in the scientific library to develop the first prototype and then to iteratively find the bottleneck in the program and improve the efficiency.

1.6 Scientific Algorithms

The multidisciplinary aspect of scientific computing is clearly visible in algorithms for scientific problems. A scientific algorithm can be either sequential or parallel. In general, algorithms in scientific software can be classified as graph algorithms, geometric algorithms, and numerical algorithms, and most scientific software calls on algorithms of all three types. Scientific software also makes use of advanced data structures and up-to-date user interfaces and graphics.

1.7 History, State-of-Art, and Perspective

The supercomputer industry started when Cray Research developed the vector-processor-powered Cray-1, in 1975. The massively parallel processing (MPP) industry emerged with the introduction of the CM-2 by Thinking Machines in 1985. Finally, 1994 brought the first Beowulf cluster, or "commidity supercomputing" (parallel computers built out of off-the-shelf components by independent vendors or do-it-your-selfers).

1.7.1 Things that are not traditional supercomputers

There have been a few successful distributed computing projects using volunteers across the internet. These projects represent the beginning attempts at "grid" computing.

- `Seti@home` uses thousands of Internet-connected computers to analyze radio telescope data in a search for extraterrestrial intelligence. The principal computation is FFT of the radio signals, and approximately 500,000 computers contribute toward a total computing effort of about 60 teraflops.

- `distributed.net` works on RSA Laboratories' RC5 cipher decryption contest and also searches for optimal Golumb rulers.

- `mersenne.org` searches for Mersenne primes, primes of the form $2^p - 1$.

- `chessbrain.net` is a distributed chess computer. On January 30, 2004, a total of 2,070 machines participated in a match against a Danish grandmaster. The game resulted in a draw by repetition.

Whereas the above examples of benevolent or harmless distributed computing, there are also other sorts of distributed computing which are frowned upon, either by the entertainment industry in the first example below, or universally in the latter two.

- Peer-to-peer file-sharing (the original Napster, followed by Gnutella, KaZaA, Freenet, and the like) can viewed as a large distributed supercomputer, although the resource being parallelized is storage rather than processing. KaZaA itself is notable because the client software contains an infamous hook (called Altnet) which allows a KaZaA corporate partner (Brilliant Digital Entertainment) to load and run arbitrary code on the client computer. Brilliant Digital has been quoted as saying they plan to use Altnet as "the next advancement in distributed bandwidth, storage and computing."[2] Altnet has so far only been used to distribute ads to KaZaA users.

- Distributed Denial of Service (DDoS) attacks harness thousands of machines (typically compromised through a security vulnerability) to attack an Internet host with so much traffic that it becomes too slow or otherwise unusable by legitimate users.

- Spam e-mailers have also increasingly turned to hundreds or thousands compromised machines to act as remailers, as a response to the practice of "blacklisting" machines thought to be spam mailers. The disturbing mode of attack is often a e-mail message containing a trojan attachment, which when executed opens a backdoor that the spammer can use to send more spam.

Although the class will not focus on these non-traditional supercomputers, the issues they have to deal with (communication between processors, load balancing, dealing with unreliable nodes) are similar to the issues that will be addressed in this class.

1.8 Analyzing the top500 List Using Excel

The following is a brief tutorial on analyzing data from the top500 website using Excel. Note that while the details provided by the top500 website will change from year to year, the ability for you to analyze this data should always be there. If the listed .xml file no longer exists, try searching the website yourself for the most current .xml file.

1.8.1 Importing the XML file

Open up Microsoft Office Excel 2003 on Windows XP. Click on the File menubar and then Open (or type `Ctrl-O`). As shown in Figure 1.1, for the the File name: text area, type in:

`http://www.top500.org/lists/2003/11/top500.200311.xml`

Click Open. A small window pops up asking how you would like to open the file. Leave the default as is and click OK (Figure 1.2).

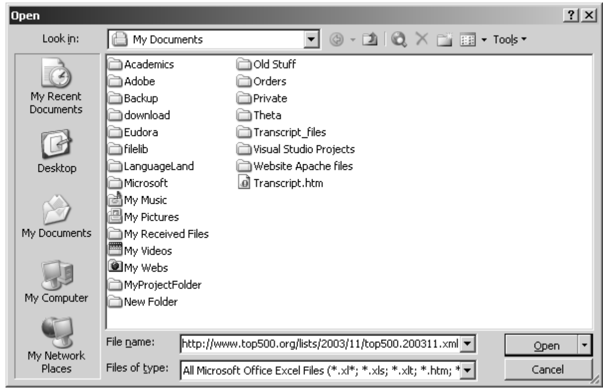


Figure 1.1: Open dialog

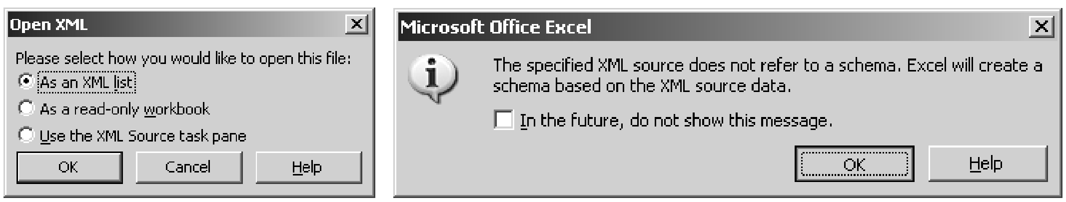


Figure 1.2: Opening the XML file

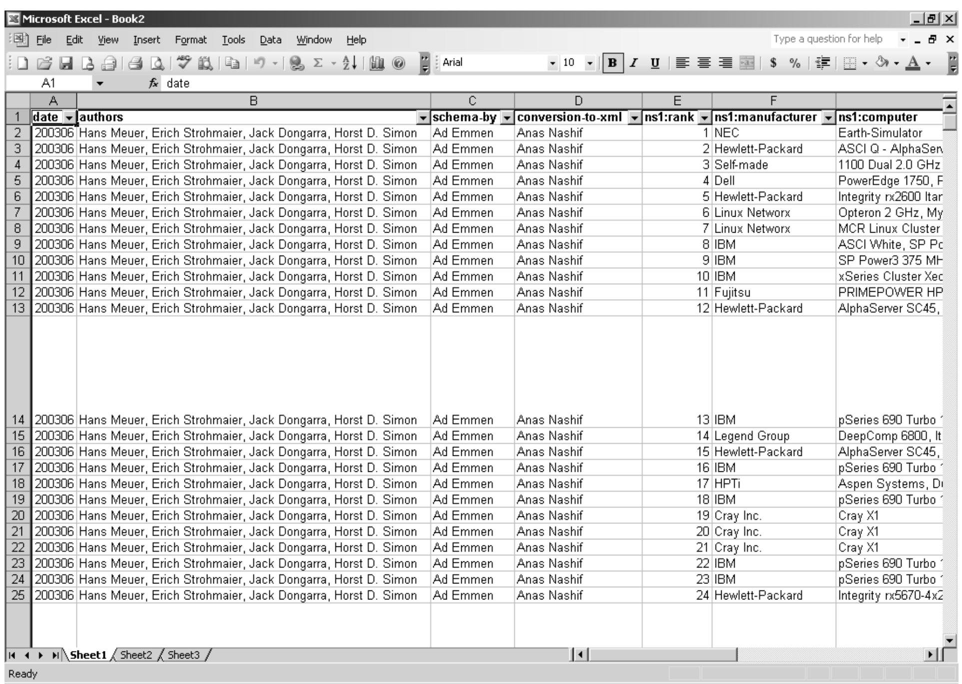
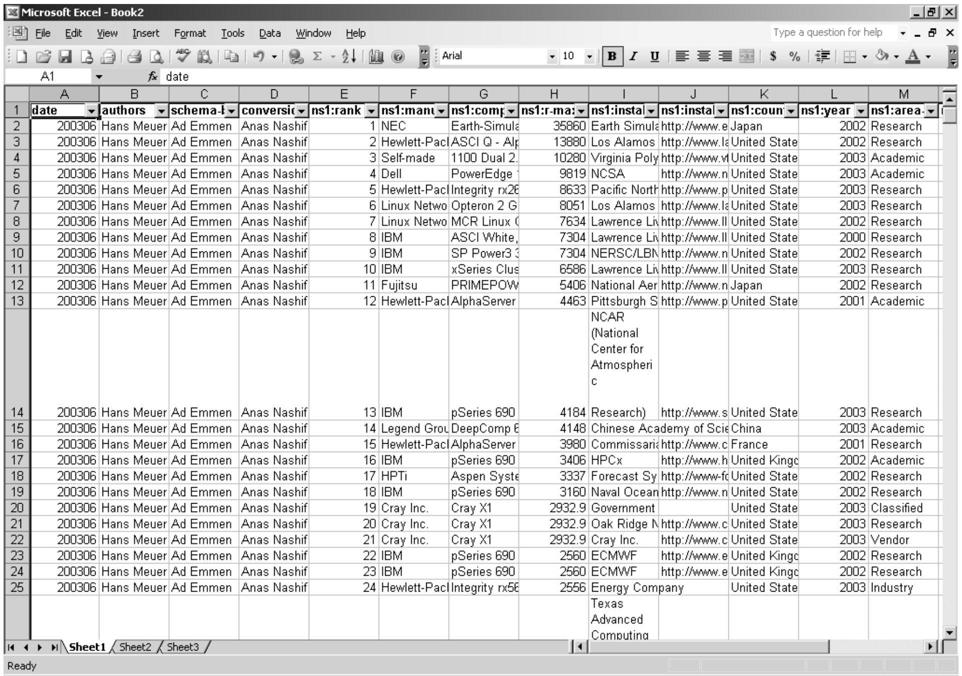


Figure 1.3: Loaded data



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	date	authors	schema-l	conversic	ns1:rank	ns1:manu	ns1:comp	ns1:r-ma:	ns1:instal	ns1:instal	ns1:coun	ns1:year	ns1:area-
2	200306	Hans Meuer	Ad Emmen	Anas Nashif	1	NEC	Earth-Simula	35860	Earth Simula	http://www.e	Japan	2002	Research
3	200306	Hans Meuer	Ad Emmen	Anas Nashif	2	Hewlett-Pacl	ASCI Q - Alp	13880	Los Alamos	http://www.la	United State	2002	Research
4	200306	Hans Meuer	Ad Emmen	Anas Nashif	3	Self-made	1100 Dual 2.	10280	Virginia Poly	http://www.vt	United State	2003	Academic
5	200306	Hans Meuer	Ad Emmen	Anas Nashif	4	Dell	PowerEdge	9819	NCSA	http://www.n	United State	2003	Academic
6	200306	Hans Meuer	Ad Emmen	Anas Nashif	5	Hewlett-Pacl	Integrity rx26	8633	Pacific North	http://www.p	United State	2003	Research
7	200306	Hans Meuer	Ad Emmen	Anas Nashif	6	Linux Netwo	Opteron 2 G	8051	Los Alamos	http://www.la	United State	2003	Research
8	200306	Hans Meuer	Ad Emmen	Anas Nashif	7	Linux Netwo	MCR Linux (	7634	Lawrence Liv	http://www.ll	United State	2002	Research
9	200306	Hans Meuer	Ad Emmen	Anas Nashif	8	IBM	ASCI White,	7304	Lawrence Liv	http://www.ll	United State	2000	Research
10	200306	Hans Meuer	Ad Emmen	Anas Nashif	9	IBM	SP Power3 3	7304	NERSC/LBN	http://www.n	United State	2002	Research
11	200306	Hans Meuer	Ad Emmen	Anas Nashif	10	IBM	xSeries Clus	6586	Lawrence Liv	http://www.ll	United State	2003	Research
12	200306	Hans Meuer	Ad Emmen	Anas Nashif	11	Fujitsu	PRIMEPOW	5406	National Aer	http://www.n	Japan	2002	Research
13	200306	Hans Meuer	Ad Emmen	Anas Nashif	12	Hewlett-Pacl	AlphaServer	4463	Pittsburgh S	http://www.p	United State	2001	Academic
14	200306	Hans Meuer	Ad Emmen	Anas Nashif	13	IBM	pSeries 690	4184	NCAR (National Center for Atmospheri c Research)	http://www.s	United State	2003	Research
15	200306	Hans Meuer	Ad Emmen	Anas Nashif	14	Legend Grou	DeepComp 6	4148	Chinese Academy of Scie		China	2003	Academic
16	200306	Hans Meuer	Ad Emmen	Anas Nashif	15	Hewlett-Pacl	AlphaServer	3980	Commissaria	http://www.c	France	2001	Research
17	200306	Hans Meuer	Ad Emmen	Anas Nashif	16	IBM	pSeries 690	3406	HPCx	http://www.h	United Kingc	2002	Academic
18	200306	Hans Meuer	Ad Emmen	Anas Nashif	17	HPTi	Aspen Syste	3337	Forecast Sy	http://www-fc	United State	2002	Research
19	200306	Hans Meuer	Ad Emmen	Anas Nashif	18	IBM	pSeries 690	3160	Naval Ocean	http://www.n	United State	2002	Research
20	200306	Hans Meuer	Ad Emmen	Anas Nashif	19	Cray Inc.	Cray X1	2932.9	Government		United State	2003	Classified
21	200306	Hans Meuer	Ad Emmen	Anas Nashif	20	Cray Inc.	Cray X1	2932.9	Oak Ridge N	http://www.c	United State	2003	Research
22	200306	Hans Meuer	Ad Emmen	Anas Nashif	21	Cray Inc.	Cray X1	2932.9	Cray Inc.	http://www.c	United State	2003	Vendor
23	200306	Hans Meuer	Ad Emmen	Anas Nashif	22	IBM	pSeries 690	2560	ECMWF	http://www.e	United Kingc	2002	Research
24	200306	Hans Meuer	Ad Emmen	Anas Nashif	23	IBM	pSeries 690	2560	ECMWF	http://www.e	United Kingc	2002	Research
25	200306	Hans Meuer	Ad Emmen	Anas Nashif	24	Hewlett-Pacl	Integrity rx56	2556	Energy Company		United State	2003	Industry
									Texas Advanced Computing				

Figure 1.4: Filtering available for each column

1.8.2 Filtering

Click OK again. Something as in Figure 1.3 should show up.

If you don't see the bolded column titles with the arrows next to them, go to the menubar and select Data → Filter → Autofilter (make sure it is checked). Type `Ctrl-A` to highlight all the entries. Go to the menubar and select Format → Column → Width. You can put in any width you'd like, but 10 would work.

You can now filter the data in numerous ways. For example, if you want to find out all SGI installations in the US, you can click on the arrow in column K (country) and select United States. Then, click on the arrow in column F (manufacturer) and select SGI. The entire data set is now sorted to those select machines (Figure 1.5).

If you continue to click on other arrows, the selection will become more and more filtered. If you would like to start over, go to the menubar and select Data → Filter → Show All. Assuming that you've started over with all the data, we can try to see all machines in both Singapore and Malaysia. Click on the arrow in column K (country) again and select (Custom...). You should see something as in Figure 1.6.

On the top right side, pull down the arrow and select Singapore. On the lower left, pull down the arrow and select equals. On the lower right, pull down the arrow and select Malaysia (Figure 1.6.

Now click OK. You should see a blank filtered list because there are no machines that are in both Singapore and Malaysia (Figure 1.7).

If you want to find out all machines in Singapore or India, you have to start off with all the data again. You perform the same thing as before except that in the Custom AutoFilter screen, you should click on the Or toggle button. You should also type in India instead of Malaysia (Figure

[2]Brilliant Digital Entertainment's Annual Report (Form 10KSB), Filed with SEC April 1, 2002.

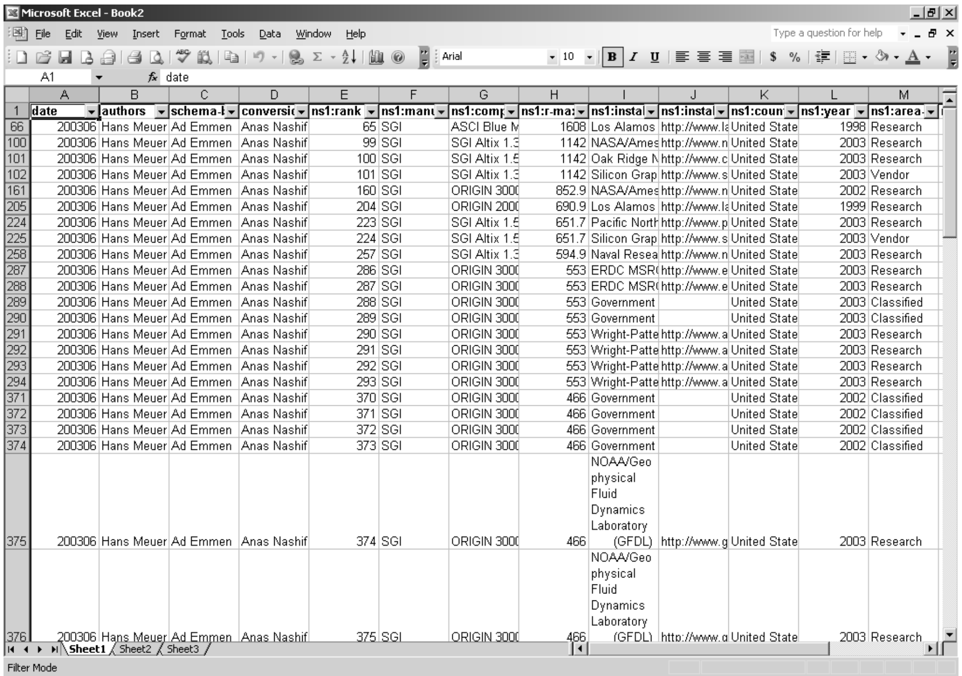


Figure 1.5: All SGI installations

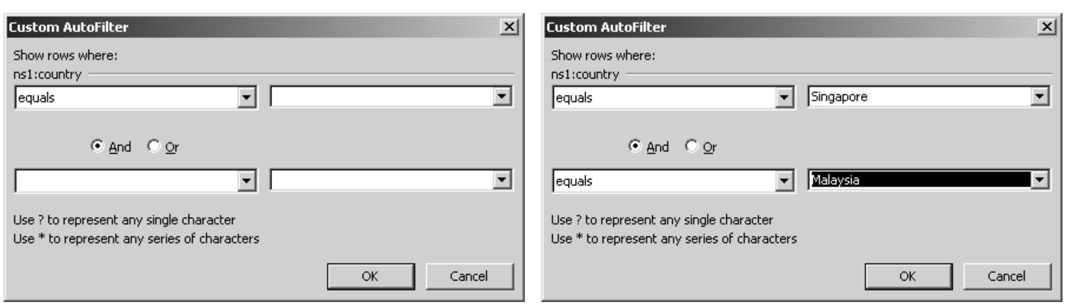


Figure 1.6: Custom filter

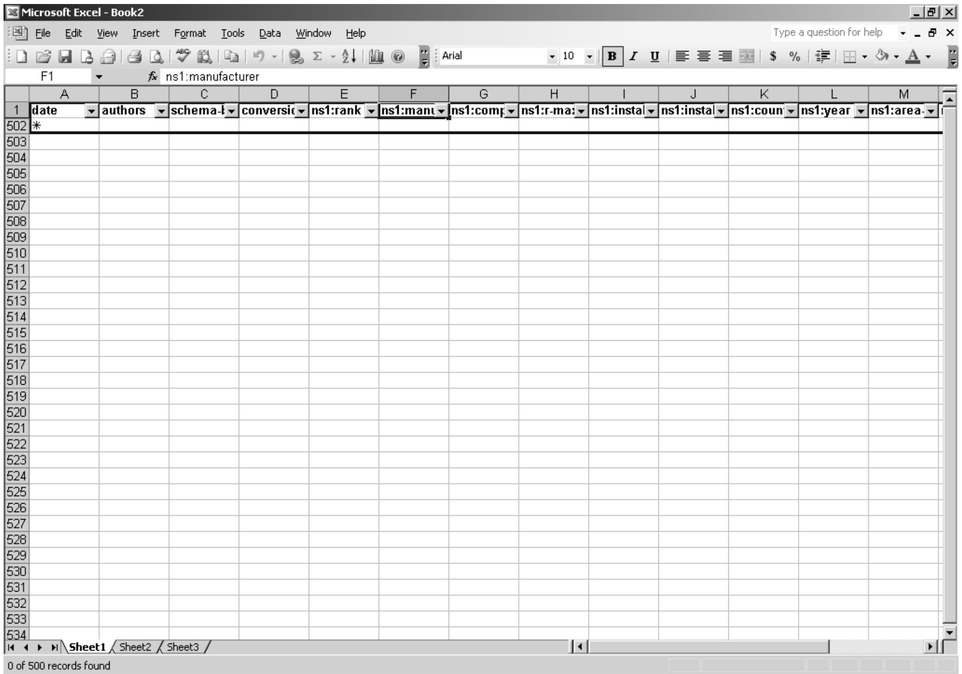


Figure 1.7: Machines in Singapore and Malaysia (none)

1.8).

Click OK and you should see something as in Figure 1.9.

1.8.3 Pivot Tables

Now let's try experimenting with a new feature called pivot tables. First, go to the menubar and click on Data → Filter → Show All to bring back all the entries. Type `Ctrl-A` to select all the entries. Go to the menubar and click on Data → PivotTable and PivotChart Report... Figure 1.10 shows the 3 wizard screens that you'll see. You can click Finish in the first screen unless you want to specify something different.

You get something as in Figure 1.11.

Let's try to find out the number of machines in each country. Scroll down the PivotTable Field List, find ns1:country, and use your mouse to drag it to where it says "Drop Row Fields Here".

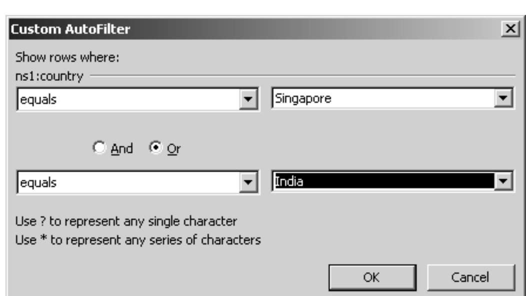


Figure 1.8: Example filter: Singapore or India

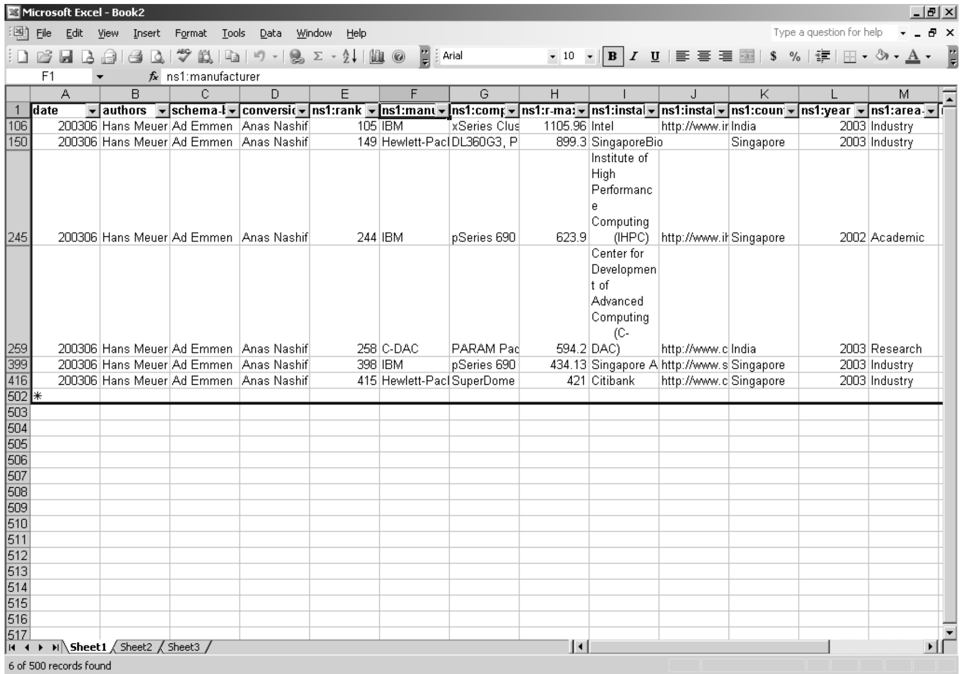


Figure 1.9: Machines in Singapore or India

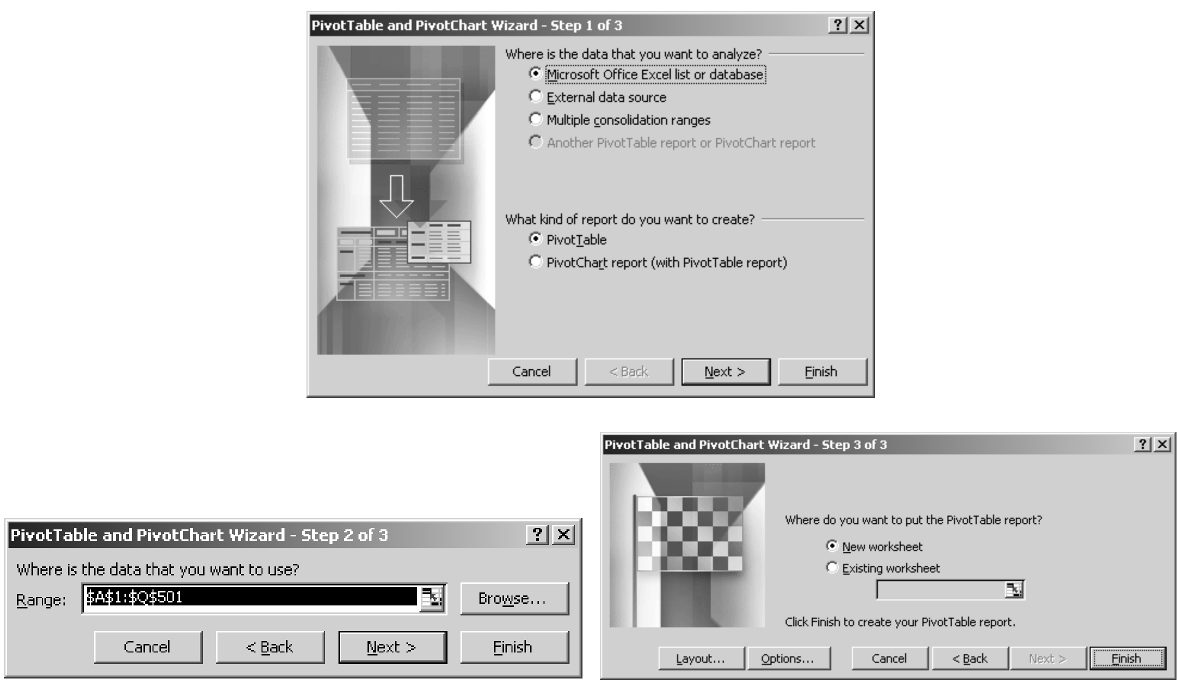


Figure 1.10: Pivot Table wizard

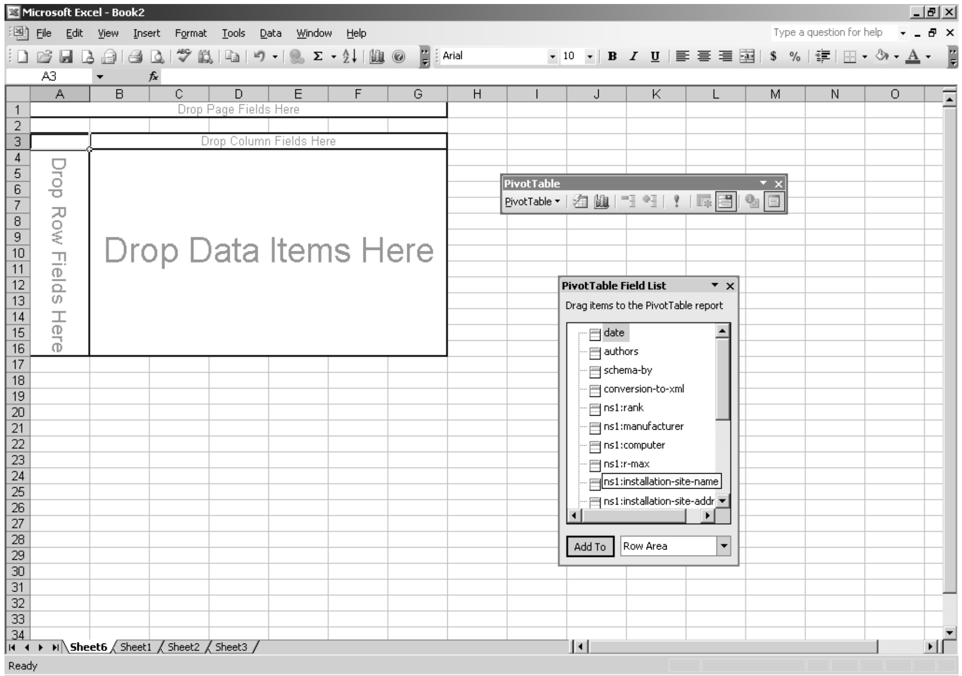


Figure 1.11: Empty Pivot Table

Now scroll down the PivotTable Field List again, find ns1:computer, and use your mouse to drag it to where it says "Drop Data Items Here" (Figure 1.12).

Notice that at the top left of the table it says "Count of ns1:computer". This means that it is counting up the number of machines for each individual country. Now let's fine the highest rank of each machine. Click on the gray ns1:country column heading and drag it back to the PivotTable Field List. Now from the PivotTable Field List, drag ns1:computer to the column where the country list was before. Click on the gray Count of ns1:computer column heading and drag it back to the PivotTable Field List. From the same list, drag ns1:rank to where the ns1:computer information was before (it says "Drop Data Items Here"). The upper left column heading should now be: Sum of ns1: rank. Double click on it and the resulting pop up is shown in Figure 1.13.

To find the highest rank of each machine, we need the data to be sorted by Min. Click on Min and click OK (Figure 1.14).

The data now shows the highest rank for each machine. Let's find the number of machines worldwide for each vendor. Following the same procedures as before, we replace the Row Fields and Data Items with ns1:manufacturer and ns1:computer, respectively. We see that the upper left column heading says "Count of ns1:computer", so we now it is finding the sum of the machines for each vendor. The screen should look like Figure 1.15.

Now let's find the minimum rank for each vendor. By now we should all be experts at this! The tricky part here is that we actually want the Max of the ranks. The screen you should get is shown in Figure 1.16.

Note that it is also possible to create 3D pivot tables. Start out with a blank slate by dragging everything back to the Pivot Table Field List. Then, from that list, drag ns1:country to where it says "Drop Row Fields Here". Drag ns1:manufacturer to where it says "Drop Column Fields Here". Now click on the arrow under the ns1:country title, click on Show All (which releases on the checks), Australia, Austria, Belarus, and Belgium. Click on the arrow under the ns1:manufacturer

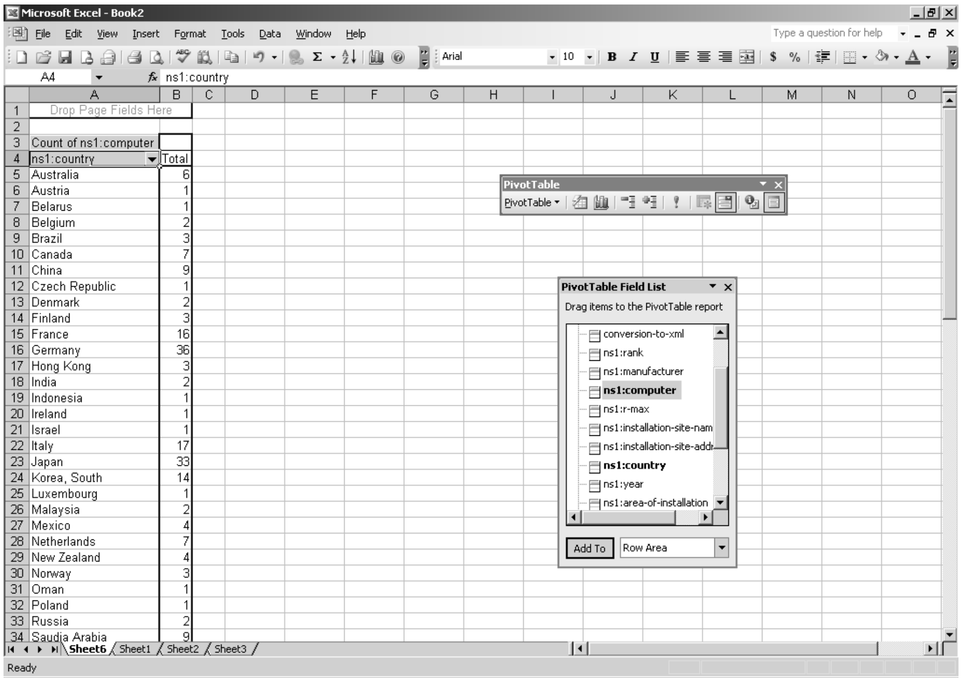


Figure 1.12: Number of machines in each country

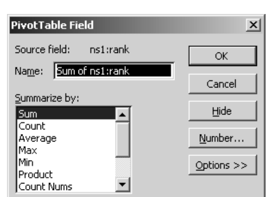


Figure 1.13: Possible functions on the rank

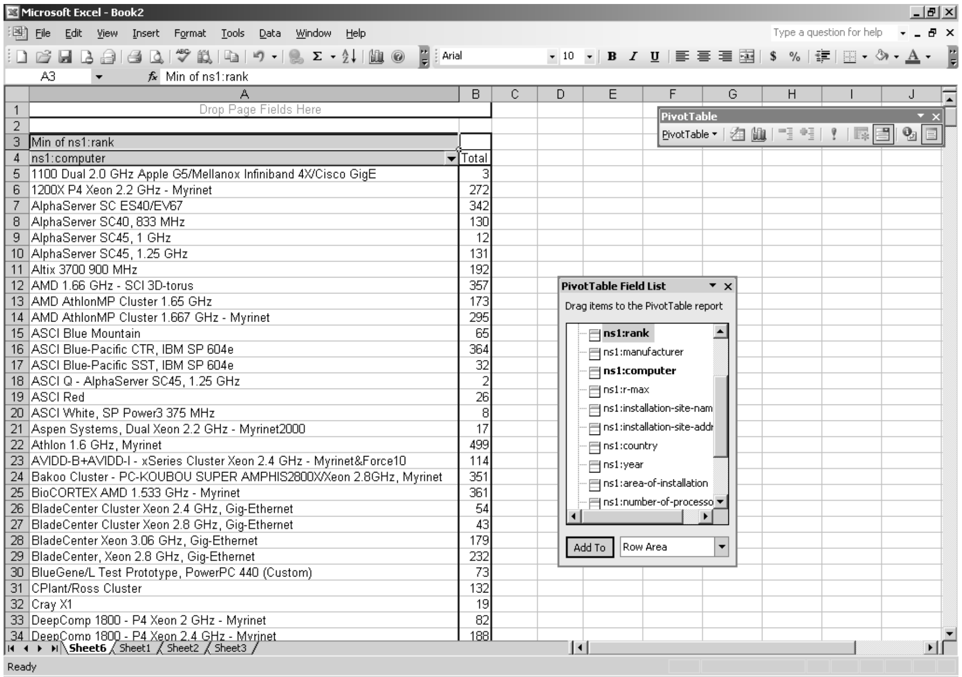


Figure 1.14: Minimum ranked machine from each country

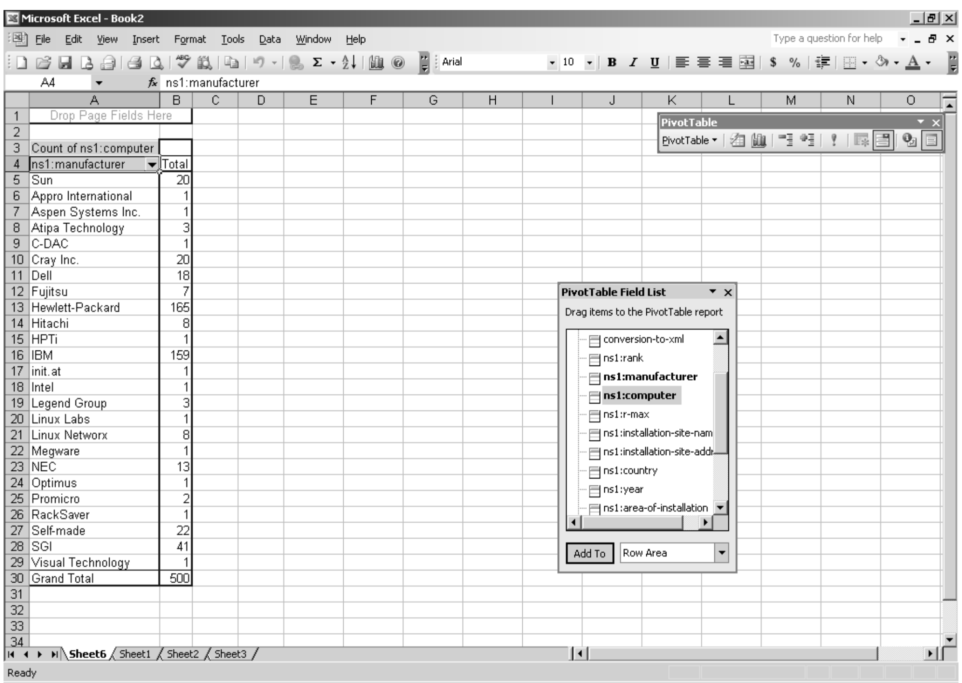


Figure 1.15: Count of machines for each vendor

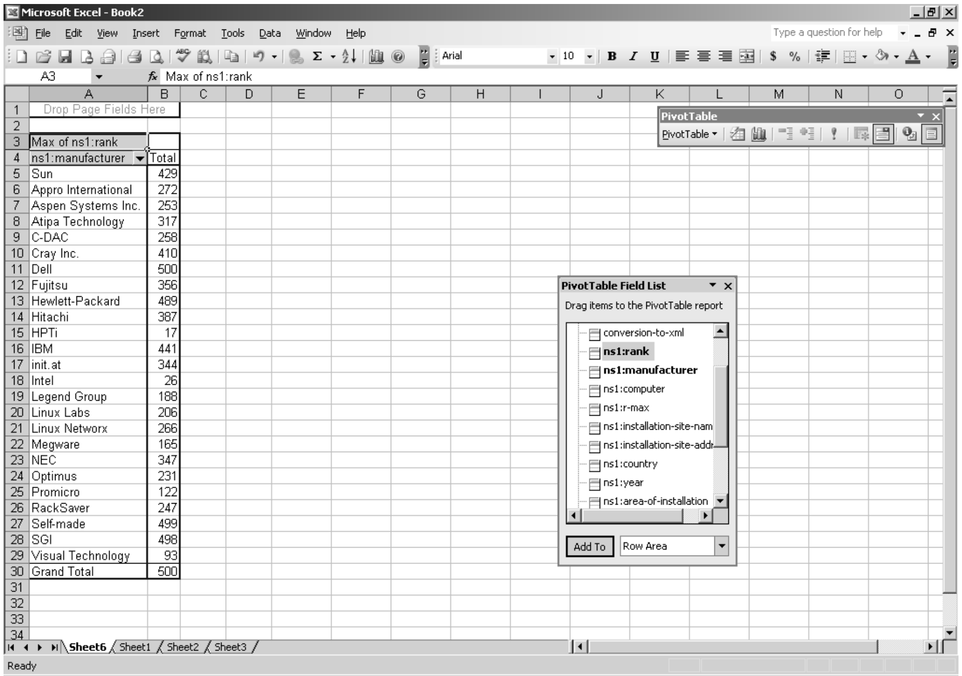


Figure 1.16: Minimum ranked machine from each vendor

title, click on Show All, Dell, Hewlett-Packward, and IBM. Drag ns1:rank from the Pivot Table Field List to where it says “Drop Data Items Here”. Double-click on the Sum of ns1:rank and select Count instead of Sum. Finally, drag ns1:year to where it says “Drop Page Fields Here”. You have successfully completed a 3D pivot table that looks like Figure 1.17.

As you can imagine, there are infinite possibilities for auto-filter and pivot table combinations. Have fun with it!

1.9 Parallel Computing: An Example

Here is an example of a problem that one can easily imagine being performed in parallel:

> **A rudimentary parallel problem:** Compute
>
> $$x_1 + x_2 + \ldots + x_P,$$
>
> where x_i is a floating point number and for purposes of exposition, let us assume P is a power of 2.

Obviously, the sum can be computed in $\log P$ (base 2 is understood) steps, simply by adding neighboring pairs recursively. (The algorithm has also been called *pairwise summation* and *cascade summation*). The data flow in this computation can be thought of as a binary tree.

(Illustrate on tree.)

Nodes represent values, either input or the result of a computation. Edges communicate values from their definition to their uses.

This is an example of what is often known as a *reduce* operation. We can replace the addition operation with any associative operator to generalize. (Actually, floating point addition is not

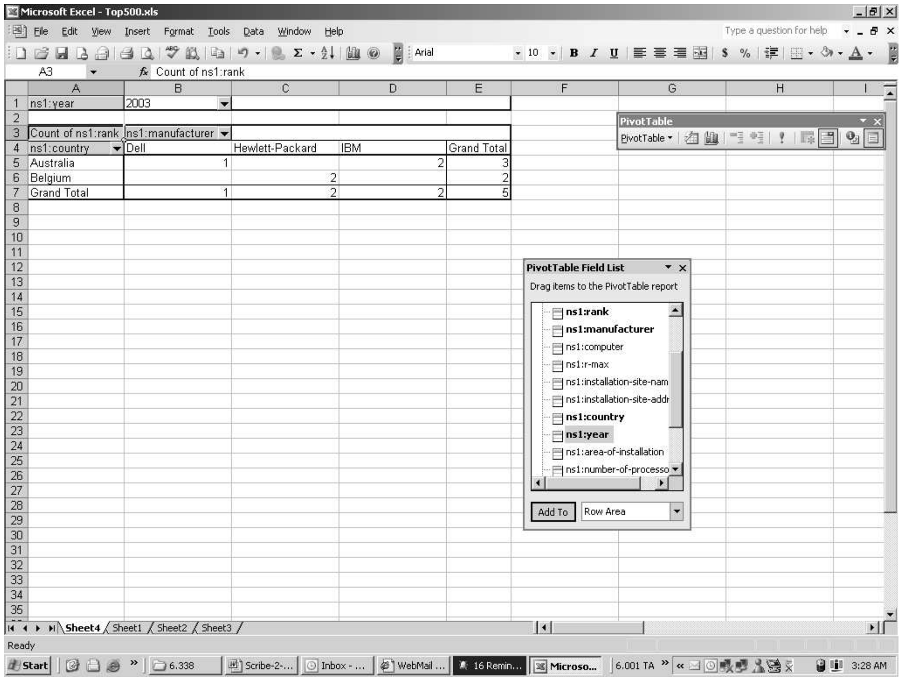


Figure 1.17: 3D pivot table

associative, leading to interesting numerical questions about the best order in which to add numbers. This is studied by Higham [49, See p. 788].)

There are so many questions though. How would you write a program to do this on P processors? Is it likely that you would want to have such a program on P processors? How would the data (the x_i) get to these processors in the first place? Would they be stored there or result from some other computation? It is far more likely that you would have $10000P$ numbers on P processors to add.

A correct parallel program is often not an efficient parallel program. A flaw in a parallel program that causes it to get the right answer *slowly* is known as a performance bug. Many beginners will write a working parallel program, obtain poor performance, and prematurely conclude that either parallelism is a bad idea, or that it is the machine that they are using which is slow.

What are the sources of performance bugs? We illustrate some of them with this little, admittedly contrived example. For this example, imagine four processors, numbered zero through three, each with its own private memory, and able to send and receive message to/from the others. As a simple approximation, assume that the time consumed by a message of size n words is $A + Bn$.

Three Bad Parallel Algorithms for Computing $1 + 2 + 3 + \cdots + 10^6$ on Four Processors and generalization

1. **Load imbalance**. One processor has too much to do, and the others sit around waiting for it. For our problem, we could eliminate all the communication by having processor zero do all the work; but we won't see any parallel speedup with this method!

2. **Excessive communication**.

   Processor zero has all the data which is divided into four equally sized parts each with a quarter of a million numbers. It ships three of the four parts to each of the other processors for proper load balancing. The results are added up on each processor, and then processor 0 adds the final four numbers together.

True, there is a tiny bit of load imbalance here, since processor zero does those few last additions. But that is nothing compared with the cost it incurs in shipping out the data to the other processors. In order to get that data out onto the network, it incurs a large cost that does not drop with the addition of more processors. (In fact, since the number of messages it sends grows like the number of processors, the time spent in the initial communication will actually increase.)

3. A **sequential bottleneck**. Let's assume the data are initially spread out among the processors; processor zero has the numbers 1 through 250,000, etc. Assume that the owner of i will add it to the running total. So there will be no load imbalance. But assume, further, that we are constrained to add the numbers in their original order! (Sounds silly when adding, but other algorithms require exactly such a constraint.) Thus, processor one may not begin its work until it receives the sum $0 + 1 + \cdots + 250,000$ from processor zero!

   We are thus requiring a sequential computation: our binary summation tree is maximally unbalanced, and has height 10^6. It is always useful to know the critical path—the length of the longest path in the dataflow graph—of the computation being parallelized. If it is excessive, change the algorithm!

   Some problems look sequential such as Fibonacci: $F_{k+1} = F_k + F_{k-1}$, but looks can be deceiving. Parallel prefix will be introduced later, which can be used to parallelize the Fibonacci computation.

1.10 Exercises

1. Compute the sum of 1 through $1,000,000$ using HPF. This amounts to a "hello world" program on whichever machine you are using. We are not currently aware of any free distributions of HPF for workstations, so your instructor will have to suggest a computer to use.

2. Download MPI to your machine and compute the sum of 1 through $1,000,000$ using C or Fortran with MPI. A number of MPI implementations may be found at `http://www.mcs.anl.gov/Projects/mpi/implementations.html` The easy to follow ten step quick start on `http://www.mcs.anl.gov/mpi/mpiinstall/node1.html#Node1` worked very easily on the MIT mathematics department's SUN network.

3. In HPF, generate $1,000,000$ real random numbers and sort them. (Use `RANDOM_NUMBER` and `GRADE_UP`.

4. (Extra Credit) Do the same in C or Fortran with MPI.

5. Set up the excessive communication situation described as the second bad parallel algorithm. Place the numbers 1 through one million in a vector on one processor. Using four processors see how quickly you can get the sum of the numbers 1 through one million.