Technological trends and their impact on the future of supercomputers

Ulrich Rüde*

Institut für Mathematik, Universität Augsburg, D-86135 Augsburg, Germany

Abstract. Semiconductor technology is advancing at an amazing rate. The changes predicted for the coming 15 years will also have a profound impact on scientific computing. Two of the most important trends are a dramatically increased potential for parallel execution within a single chip, and a serious gap between the speed of memory and the processor. Future computer architectures will force the developers of scientific computing applications to address problems originating from fine grain parallelism and problems caused by the memory bottleneck.

1 Introduction

Scientific computing applications have a seemingly unlimited demand for computing resources, since with faster processors and larger memory more complex models can be simulated and the accuracy of simulations can be improved. Scientific computing is thus driving the development of *supercomputers*, however, the architecture of supercomputers is also driving scientific computing research. In the 1980ies the speed advantage of vector computers has caused research into the vectorization of scientific computing codes, and in the 1990ies we see increasing efforts to parallelize the most demanding applications for parallel architectures. Such research is motivated by trends in computer architecture.

Because of the enormous development efforts being made, mass market microprocessors have been closing the performance gap to special purpose processors quickly. In fact, the world's most powerful supercomputer today is the *ASCI Red[2]*, a massively parallel machine built from 9152 Pentium Pro processors. The same processors can also be found in many PC based servers. In principle, we might be able to construct processors much more suitable for scientific computing than Pentiums, but the development cost could not be amortized by the comparatively small scientific computing market. So, obviously, the shift to commodity processors for supercomputers is caused by market forces.

The development of computer technology is empirically governed by *Moore's* law that can be interpreted to say that the speed of computers doubles every 18 months. Of course the development of ever faster processors must have physical limits and thus Moore's law cannot hold forever. Surprisingly, however, the progress in semiconductor technology for the next 10–15 years still

* Partially supported by DFG grant RU 422/7-1

seems to be quite predictable. For this time span, fundamental physical limits will not yet slow down the progress significantly. All foreseeable problems for the next 1000 fold increase in performance are of only technical nature. In this article we will attempt to draw conclusions for the future of supercomputing based on the predictions for semiconductor technology from the *National Technology Roadmap for Semiconductors* [11].

For a more detailed discussion of future computer architecture issues, see e.g. the article by Burger and Goodman on [3] and the other articles in the September 1997 special issue of the IEEE Computer Journal on *Billion Transistor Architectures*.

In this current proceedings there are also articles concerned with performance issues related to computer architecture, for example S. Turek's design of a next generation computational fluid dynamics package [12] is strongly influenced by computer architecture considerations, and similarly, K. Gärtner's concepts for the efficient simulation of semiconductors [5] are based on a careful study of real life machine performance.

At the beginning of this article, it may be necessary to make the disclaimer: Futurology is notoriously unreliable. Any predictions, based on an quantitative extrapolation of current trends can of course be overthrown by unforeseen qualitative changes. Our projections are limited to a time span of 10-15 years as it is covered by the present semiconductor roadmap. Hopefully, this will improve the chances of this article to hit a bit of the truth, however, only when the future has become the present, we will know for sure.

All discussion in this article is is based on publicly available information, gathered mostly by browsing the internet: information provided on manufacturer's WWW sites, press announcements, slides of talks from various people, technical reports from FTP servers, arguments from various news groups. Much of the information is short-lived and paper documents are often outdated as soon as they have been printed. When you hold the printed version of this in your hands, maybe a good idea will be to check the author's WWW site at `scicomp.math.uni-augsburg.de/ruede/me.html` to check for newer material.

2 State of the art in Microprocessors

Current[1] high end microprocessors, like the Digital Alpha 21164 operate at clock rates of more than 600 MHz and deliver a peak performance of 1.2×10^9 floating point operations per second (1.2 GFlop). Such computational power is the result of a dramatic technological development over the past four decades that has produced ever faster and more powerful computers.

The improvements in computers are driven by the integration of smaller and smaller structures in silicon. Today, the state of the art is a 0.25μm process, permitting integrated circuits with more than 10^7 transistors on a

[1] This is being written in May 1998

single chip. Single chip microprocessors are thus amazingly complex devices. Their computing power relies partly on fast operation of individual circuits. As a direct consequence of miniaturization, clock rates below 2 nanosecond (ns, 1 ns $= 10^{-9}$ seconds) can be achieved and there is no doubt that switching times below 1 ns will be possible within a few years.

2.1 Trend: Increasing on-chip parallelism

However, fast clock rates alone would not explain the performance of present microprocessors. The other factor contributing to the computational power is the *architecture*, that is the design of the processors themselves. Current processors exploit the abundance of functionality that can be realized with millions of transistors in complex designs, by using *parallelism on the chip*. All modern chips employ pipelining combined with the superscalar execution of machine instructions.

However, it is important to realize that the present state is just a transition, and that the evolution continues. The current *Roadmap for Semiconductors* [11] predicts integrated circuits with 1.4 billion (1.4×10^9) transistors and clock rates of 10 GHz for the year 2012. An excerpt from the roadmap for microprocessors is presented in Table 1. If such processors will become reality, the consequences are amazing. Even a simple argument can show that in principle, more than the full aggregate processing power of a 1000 processor Cray T3E computer, currently rated as the second fastest computer in the world [4], could be implemented on a single chip of the 2012 generation.

1997 National Technology Roadmap for Semiconductors							
Year of 1st shipment	1997	1999	2001	2003	2006	2009	2012
Local clock (Mhz)	750	1250	1500	2100	3500	6000	10000
Across chip (Mhz)	750	1200	1400	1600	2000	2500	3000
Chip size (mm^2)	300	340	385	430	520	620	750
Feature size (nm)	250	180	150	130	100	70	50
Number of chip I/O	1450	2000	2400	3000	4000	5400	7300
Transistors/chip	11M	21M	40M	76M	200M	520M	1.4B

Table 1. Excerpt from the 1997 semiconductor roadmap

The Cray T3E uses Alpha 21164 processors, each of which contains 10 million transistors. In 2012, 1 billion transistors will be integrated in a single chip and thus 100 copies of an Alpha processor could be arranged on such a chip. This would result in a 100-way-parallel one-chip multiprocessor. Besides the on-chip parallel execution, a speedup by a factor of 20 would occur by the clock rate increase from the current 500 MHz to 10 GHz. Thus this hypothetical piece of silicon in 2012 would deliver a peak performance of 2 Tflop (2×10^{12} Flop). Of course it seems unlikely that a chip with this configuration will ever be built. However, the argument clearly illustrates the potential performance increase that will become possible by 2012.

2.2 Possible future CPUs

According to the roadmap [11], semiconductor technology is likely to advance with unbroken speed. However, as in the past, only the clock rate improvements will result in a performance gain for free. Another part of the potential performance gain should be caused by the increase in the number of transistors by a factor 100 which would permit a 100-fold increased of the on chip parallelism. Unfortunately, however, the concepts used in present microprocessors to exploit instruction level parallelism do not seem to scale by another factor 100.

Current CPUs rely on the compile time arrangement of instructions for pipelining and superscalar execution. Additionally, current CPUs may employ out-of-order execution at run time. However, the majority of current programs exhibit just a limited amount of compiler or hardware detectable parallelism, and it seems impossible that the current mechanisms will suffice to exploit hyperscalar hyperpipelined CPUs with a 100 fold increase in instruction level parallelism. Future increases of on-chip parallelism will increasingly require explicitly adapted codes. Thus we should expect that many of our programs need to be rewritten to efficiently exploit future CPU architectures.

This is our first conclusion for scientific computing: Up to now we have been exploiting instruction level parallelism without much effort because the compilers and hardware automatically did this kind of parallelization for us. In the future this may not be possible anymore, and we may have to help the system to detect the parallelism in our applications. This may require the use of languages and programming styles that make the fine grain parallelism more explicit. How much and in which form this on-chip parallelism will be available to us, will depend on how the CPU architecture evolves. But it should be clear that potentially, the performance gain is as large as the speedups we obtain from our current efforts for massively parallel systems. The main difference may be that if we learn to use the instruction level parallelism, it will be a gain essentially free of cost. If we do not use it, we will simply waste a large portion of the future CPU circuit complexity.

3 The von-Neumann bottleneck: memory performance

Besides the problem of using instruction level parallelism, there may be an equally serious problem ahead of us. At one GFlop speed, an Alpha 21164 processor consumes 2 operands and produces one result in each nanosecond. Assuming 8 Byte words, the total bandwidth required to sustain the peak performance is 24 GByte per second. In contrast to this, typical memory systems built around an Alpha processor have a nominal peak bandwidth of less than 1 GByte/second, and benchmarks [9] demonstrate that in practice even less than that is really usable.

Thus, in many applications, modern high speed processors cannot reach their peak performance, since the data cannot be transported quickly enough from the microprocessor to the memory system and vice versa. Why this is the case, and how this will extrapolate to the future is the next major of our discussion.

3.1 Memory systems and memory interfaces

Parallel to the evolution of microprocessors, the development of dynamic random access memory (DRAM) has seen amazing progress. The DRAM process permits an even denser arrangement of structures on the silicon than on logic chips. Current DRAM integrates 64 or 256 million bits on a single chip. However, while the density and thus the storage capacity have been improved tremendously, the access times have improved only slowly.

Latency issues. This creates a *latency* problem. At 500 Mhz, a two-way superscalar CPU needs a set of 4 operands each 2 ns. Typical external memory systems, however, require more than 100 ns to deliver a word from main memory to the CPU. The latency of a DRAM memory system is therefore typically a factor of 50 slower than the clock rate of the processor. There are several ways to deal with this *latency gap*.

A technique that has been successful in scientific computing applications is *vectorization* combined with memory interleaving. The second technique to deal with latencies are *caches*, as they are used in virtually all modern microprocessors. Faster memory can be built by using more expensive technology. Static random access memory (SRAM) is faster than DRAM but also significantly more expensive. Memory may also be integrated into the processor chip itself, but transistors on the processor chip are an even more precious resource. Therefore, SRAM cache memory off-chip is typically 2 orders of magnitude smaller than the main memory, and the on chip caches are another order of magnitude smaller than off chip caches. (See, however, the design of the HP PA 8500 chip [8]).

In general, the concept of a flat memory where each memory cell can be accessed in the same time is not a realistic model of computers that employ multiple layers of caches. With two or even three layers of caches plus the registers, we should regard many modern machines as having a *hierarchical memory organization*.

The third technique to deal with (memory) latencies is *execution out of order*. Here an instruction may be suspended waiting for the data while other operations are being performed. This idea is taken to an extreme in *multi-threaded architectures* (see [1]), where the program is broken up in many independent execution threads. Each of these threads can be suspended waiting for data while other threads whose data is available can be executed.

Many present designs use combinations of the above principles to deal with latency problems, and when the gap between processor and memory latency opens further, such techniques may even become important within the chip itself.

Bandwidth issues. So far we have discussed techniques to hide or tolerate the latency of memory access. However, there is also a problem with *bandwidth*. As argued above, the typical memory systems provide just a fraction of the bandwidth required by current microprocessors. These processors can only come close to their peak performance, when the data is almost exclusively accessed from the faster levels in the memory hierarchy.

The limited bandwidth of microprocessor based machines is in fact the only reason why classical (vector) supercomputers still have a significant performance advantages in some applications. Even though their peak performance is not so much superior to microprocessors, their bandwidth makes them preferable to cache based machines in applications where the caches are not effective. This is the case for many scientific computing programs.

With the memory architecture found in vector supercomputers, it is possible to construct memory systems with up to 4 TByte/sec overall bandwidth as e.g. in the NEC SX 4-32 [10]. However, all such designs rely on physically distributed CPUs, where multiple vector pipelines are implemented in separate chips, each having an independent data path to the memory system. The disadvantage of such involved memory architecture is its cost.

3.2 How much performance fits on a single chip

Besides its good efficiency in scientific computing applications, the vector architecture is attractive, since it is a simple design. A vector unit with both a multiplication and addition pipeline may be implemented with approximately 500,000 transistors. Furthermore, a vector pipeline can be operated at high clock rates. With 10 million transistors available on a chip, we could arrange 40 such pipelines on the die, and would thus have a chip with 40 GFlop peak performance at 500 MHz. It is interesting to compare this with a NEC SX-4 which in maximal configuration has a total of 256 vector pipelines organized in 32 processors. Each pipeline of the SX-4 is running at 125 Mhz, resulting in an aggregate peak performance of 64 GFlop. Thus, the computational performance of a single chip could be on the order of a full sized vector supercomputer if we did not have a problem with transporting data to and from the chip.

If it were for the floating point peak performance alone, the power of a present vector supercomputer should be available at no higher cost than that of a standard microprocessor. In fact, each single workstation or PC could have the computing power of a multi-million Dollar vector supercomputer. What makes vector supercomputers more expensive is their memory bandwidth. Thus, the computational scientist who learns to live with limited bandwidth by making his or her programs execute with perfect data locality, should be able to buy the computational power of a full sized vector supercomputer at the price of a single PC.

Of course it will be impossible in practice to restructure all of our algorithms in such an extreme way. However, improving the data locality of our programs is a cost effective way to improve the efficiency of our algorithms.

4 Balancing processor speed with memory bandwidth in the future

This has been a description of the situation today. However, the semiconductor roadmap predicts the possibility of 2000 times more powerful chips within 15 years. Let us discuss such chips on the basis of current microprocessors, like the Alpha 21164 with 1 GFlop peak performance and a corresponding read bandwidth requirement of 16 GByte per second. Scaling the peak performance up by a factor of 2000 in the year 2012, such a processor would require 2000 times the bandwidth. Thus 32 TByte data per second would have to be read from memory into a single chip.

On the other hand, the semiconductor roadmap [11] suggests that a chip in this generation may have some 7000 pads connecting it to the outside world. Even if we assume that 4096 of these were used for data input only, the bus would still have to be operated at 62.5 GHz to provide sufficient bandwidth. This is much beyond anything imaginable in the time frame considered. Note that using 1 billion transistors for vector pipelines would result in an even more ridiculous misbalance between memory and processor performance.

This simple estimate leads to the conclusion that we can either decide not to fully exploit our future on-chip transistor budget for processing, and thus be content with lower performance processors, or we will have to live with processors whose bandwidth requirement cannot be satisfied from outside memory. To summarize, there are three main ways out of the bandwidth dilemma:

1. Some yet unknown innovation will provide sufficient bandwidth from a piece of silicon to the outside world.
2. We may simply not build chips with that much computing power, (just as we have decided not to put 40 vector pipelines on a single chip today). The question then is, what else will we do with the transistor budget that the technology of 2012 will give us.
3. We will invent computer architectures and programming techniques that permit us to live with insufficient off-chip bandwidth.

From the present perspective, it seems unlikely that solution number 1 is realistic, though of course we can always keep hoping for a miracle. There will be an increased pressure to develop alternative technologies and that may significantly speed up innovation.

The main potential for a solution seems to come from a combination of approach 2 and 3. It seems likely that an increasing fraction of the chip area will be devoted to making off chip bandwidth unnecessary, or — phrased explicitly — processing power is traded for an increasing fraction of on-chip memory. Maybe this memory will still take the form of caches, but maybe we will also see alternative design principles.

Additionally, it will be necessary for programmers to become conscious of the limited off-chip bandwidth. Even if we decide to put more memory on chip, it seems impossible that all system memory can be held in a single chip. Note that even today, a 1 GFlop processor should have 128 MByte to 1 GByte for serious large scale scientific computing problems. However, even with 1 billion transistors, just about 20 MByte of on chip SRAM could be implemented. This is by far not enough to get away without off-chip memory.

There have been suggestions to use a part of the chip area not for SRAM, but for implementing DRAM to obtain a denser packaging of the on-chip memory. The most prominent proponent of this idea is D. Patterson with his IRAM project [7]. At the time when 1 billion transistor microprocessors become possible, the corresponding projections for DRAM would permit the construction of 8-32 GByte DRAM on the same die area. Technologically, it will not be completely trivial to integrate a DRAM and a logic part on the same chip, but let us assume that it can be done. Unfortunately, even 32 GByte will likely be considered a small memory when compared with the processing power that we could implement.

There are two extreme configurations that we could design with an IRAM approach.

In the first approach, almost all of the chip is used for DRAM, only a small portion is occupied by a processing unit. Assuming clock rates of 10 GHz, even a processor core with 10 million transistors should deliver 20 GFlop and would thus be approximately balanced by 16 GByte of memory. So we might design the processing unit similar to an Alpha 21164 today, only that it could run at 10 GHz. The processor logic would occupy about 1% of the die area, and the remaining 99 % were used to implement between 8 and 32 GByte of DRAM. The unique feature of such a design would be that no conventional memory is required. All system memory is on the IRAM chips.

Such a device might be interesting as a single chip computer for embedded applications, but for engineering (workstation-type) applications we might demand a significantly higher performance by the year 2012. Systems with a nominal performance of 200–2000 GFlop could be built cost-effectively by assembling systems consisting of 10–100 of these IRAM chips, but of course they would be parallel machines. Their use would already require a substantial effort to exploit the parallelism and would force a change in programming paradigm towards explicit parallel computing even for desktop applications.

Alternatively, we may try to accelerate the processor on the chip as much as possible, using a major portion of available chip area for increasing the processing power. The portion devoted to on-chip memory may still be larger than today, but we would think of processor cores with hundreds of millions of transistors. At 10 GHz, we might thus expect these processors to perform at 200–500 GFlop which may be the expected performance of an engineering workstation in that technology. Such a performance would require hundreds of GBytes of memory to be useful and this makes an off-chip memory system necessary. As argued above, the technological trends do not suggest that this off-chip memory could be driven with sufficient bandwidth.

No matter how on-chip and off-chip memory would be organized, programmers will have to learn that reasonable performance of such a system could only be expected with programs being aware of the reduced performance of the off-chip memory access.

Note that the question of data locality occurs in both approaches. In the first model, each processor will have sufficient bandwidth within its local on-chip memory, but the solution of a problem distributed among many such processors would require a data allocation such that each IRAM processor mainly only used its own local memory. In the second model, the processor would have access to a fast cache-like on-chip memory of considerable size, but programs would have to be organized such that data brought to the on-chip memory were re-used as much as possible.

For the algorithmic development, *data locality* may therefore become a prime issue. In general, the cost of floating point *operations* is quickly decreasing. Moving data is what makes computing expensive. This is the rationale of an ongoing project directed at improving the data locality of elliptic PDE solvers [6]

5 The impact on scientific computing algorithms

What of all this technology discussion is of relevance for scientific computing? Should not primarily computer architects, microprocessor designers, compiler writers and operating system developers worry about these problems?

Unfortunately, it seems that these issues will have an impact on how we evaluate and design scientific computing algorithms. Additionally it may be important that we analyze the potential impact of these trends on our algorithms to give the computer architects correct guidelines on which requirements our problems have.

For exposition, we have selected a simple model problem. Scalar elliptic partial differential equations can be solved by a variety of solution algorithms. We will focus on two simple algorithmic alternatives.

Assume that the discretization results in a linear system with n^2 unknowns (corresponding to a $n \times n$ mesh) with 9 entries in each row (corresponding to 9 point difference stencil or bilinear finite elements). Assuming a setup as a banded system, the equations may be solved by a banded Gaussian elimination in about $4n^4$ operations, using about $2n^3$ words of memory, since fill-in partially destroys the sparsity structure.

This algorithm is certainly not optimal for the problem. In the class of direct methods, e.g. a nested dissection provides better complexity. Theoretically, the fastest known algorithms are iterative multigrid methods. However, because of its relative simplicity, we will refer to banded Gaussian elimination as the representative for the class of direct methods.

As alternative, we will study the successive over-relaxation (SOR) algorithm. We assume that the optimal relaxation parameter is known and that we obtain a sufficiently accurate result after n iterations. Thus the computational work is $18n^3$ operations using $11n^2$ words of memory ($9n$ for the matrix, and each n for the solution and right hand side). In computational complexity, the SOR method would thus be considered as clearly preferable over banded elimination.

However, let us now consider a specific machine architecture, with memory bandwidth limitations, similar to current high performance PCs. Let us assume that by perfect pipelining latency is never an issue, but that main memory only supports 200 MByte/second of memory bandwidth, while a single level of on-chip cache provides unlimited bandwidth in the sense that the machine peak computational performance of 1 GFlop is reached.

Now assume that the elliptic equation problem has size $n = 100$, and the cache is 100 KByte large. The data set for the full problem requires 880 KByte and is too large to fit in the cache. Thus the SOR computation is bandwidth limited. Using the simplified assumption that the 880 KByte must be read in each of the n iterations, we can easily find that the computing time will be 440 msec which corresponds to a performance of about 41 MFlop only. With no memory bandwidth restrictions and 1 GFlop performance, the $18n^3$ operations should only take 18 msec.

For the banded elimination algorithm, a closer look reveals that the *elimination front* is 100 elements large, so that the processing of the front will fit in the cache. Thus the $4n^4$ operations can be performed without memory bandwidth limitations, resulting in a time of 400 msec . In the Gauss-algorithm, an LR factorization of the original matrix is computed. The R factor must be stored for use in the subsequent back substitution. We assume that both writing and reading of the n^3 words of the matrix R requires 40 msec each and that the write operation can be completely overlapped with the computation of R, so that only an additional 40 msec have to be added for reading the R factor in the back substitution. The total time will thus be $440msec$ and thus be exactly as much as for the SOR method.

We may next consider what changes in the architecture would benefit which algorithm. Improving the memory bandwidth would speed up the SOR algorithm and give it a advantage compared to elimination. This, however may be quite expensive, requiring wider buses, higher bus clock rates, etc.

Alternatively, we might consider chips with increased computational performance. This change would have no effect on the run time of SOR, but might significantly speed up the elimination algorithm. If the chip delivered 10 GFlop instead of 1, the elimination could be performed in just 40 ms, and — fully overlapped with writing the R factor — we would obtain a total solution time of 80 msec. To obtain the same time with SOR, the bandwidth would have to be increased to 2 GByte/sec.

From the cost perspective, it is not clear whether adding 4 million transistors to the chip (the cost of 8 addition and 8 multiplication pipelines at 500 Mhz) were not relatively cheaper than building a 10 times more powerful memory system.

We should mention that the data locality of the SOR algorithm can be improved by suitably restructuring the loops, but that, in contrast to the caching of the elimination front in the banded elimination, this presently requires explicit and tedious programming.

Of course one may argue that we have presented a problem with fixed size, and that when n grows, the algorithm with better asymptotic complexity will always win out. This is definitely correct, especially since for larger n, the elimination front will eventually become larger than the on chip memory and its processing will then also be subjected to bandwidth limitations. It is also clear that for three dimensional partial differential equations the situation is completely different.

However, let us consider the above situation in scaled-up scenario, corresponding to the situation that we may have in 15 years form now. Since we expect a performance improvement by at least a factor 1000 in that time span, let us demand that a problem with $n = 1000$ can be solved in the same time as the $n = 100$ problem can be solved today. For a $O(n^3)$ algorithm, like SOR, a scale up from $n = 100$ to $n = 1000$ with constant compute time seems to be reasonable. However, while we can assume that the 1000 fold increase in MFlop from 41 at present to 41 GFlop in 2012 will be easy to accomplish in principle, we must also scale up the bandwidth by a factor of 1000 to 200 GByte/second. This means that a 1024 bit memory bus will have to be operated at 1.6 GHz.

For scaling a banded solver, we first need more on chip memory, but the 10^2 times increase in on chip memory size to 10 MByte should be no problem in a billion transistor architecture. The computational performance seems to be more difficult to achieve: To scale up in constant time, a 10^4-times increase in peak performance is necessary. This can only partly be accomplished by the 20-times increase in clock frequency from 500 MHz at present to 10 GHz in 2012. The remaining factor of 50 would have to come from using 50 Add/Mult pipelines in parallel. Additionally the pipeleines must be connected them to the on-chip memory with sufficient bandwidth. From our present perspective this may seem to be impossible, however, if each of the Add/Mult pipelines were implemented with 0.5 million transistors, they would just require a total of 25 million transistors and thus just 2.5 % of the available on chip-logic in

2012. Thus, it seems to be quite possible that we could implement this 1 TFlop of performance within a single chip without much trouble and thus the banded solver it is again unclear whether this approach were not more cost effective than an equally fast SOR method.

Summing up this discussion, we have demonstrated that algorithmic questions are nontrivially affected by the growing gap between processor and memory performance. Traditional estimates based on the work count alone do not give a realistic predictions of the run time of programs for current memory systems. The analysis of algorithms and computing equipment will increasingly also have to deal with memory performance issues.

In general, these issues may have even have an impact on which algorithms we chose for a given application, and in turn the algorithmic choice will determine which computer architectures are most suited. Once we have accepted without historical prejudices that computation is almost free, while memory access is expensive, we may be lead to unconventional algorithmic approaches.

6 Conclusion

In conclusion (and in contrast to the mainstream belief), we do not see any evidence that the progress in computing during the coming 15 years will be limited by the physical constraints in semiconductor technology, but in a lack of appropriate designs for the architecture of computers and the software that we use. Present concepts in computer architecture and the fundamental building blocks of numerical software do not seem to scale well to what the evolution of technology will make possible. Instead of hitting physical limits in the design of more powerful computers, we may first find that the further progress in scientific computing is limited by a fundamental misconception in the theory of algorithmic complexity. We will have to learn that not *processing* data is costly, but *moving* data. For the next decade, we therefore expect that we will have to rethink the basic assumptions behind computing much more than we had in the past.

Software typically has a much longer lifespan than hardware. Therefore program development today should as much as possible be aware of these issues and should be based on concepts, techniques, and tools that can scale into the next century.

References

References

1. R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith. The Tera computer system. Can be downloaded from URL: http://www.tera.com/web/library-psdocs.html.

2. ASCI Red Teraflops Supercomputer. Brochure accessible from URL: http://www.llnl.gov/asci/overview/.
3. D. Burger and J. R. Goodman. Billion-transistor architectures - guest editors' introduction. *IEEE Computer*, 30(9), September 1997.
4. J. Dongarra, H.-W. Meuer, and E. Strohmeier. TOP500 supercomputer sites, 1997. Can be accessed from URL: http://www.top500.org/.
5. K. Gärtner. Parallel multigrid methods for the continuity equations in semiconductor device simulation, 1998. To be publsihed in this proceddings.
6. H. Hellwagner, M. Kowarschik, U. Rüde, and C. Weiß. Data local iterative methods for the efficient solution of partial differential equations, 1998. http://wwwbode.informatik.tu-muenchen.de/Par/arch/cache/index.html.
7. C. E. Kozyrakis, S. Perissakis, D. A. Patterson, T. E. Anderson, K. Asanovic, N. Cardwell, R. Fromm, J. Golbus, B. Gribstad, K. Keeton, R. Thomas, N. Treuhaft, and K. A. Yelick. Scalable processors in the billion-transistor era: Iram. *IEEE Computer*, 30(9), September 1997.
8. G. Lesartre and D. Hunt. PA-8500: The continuing evolution of the PA-8000 family, 1997. http://www.hp.com/computing/framed/technology/micropro/pa-8500/docs/8500.html.
9. J. McCalpin. Stream: Measuring sustainable memory bandwidth in high performance computers. http://www.cs.virginia.edu/stream/.
10. NEC SX-4. Data obtained from URL: http://www.hlrs.de/hpc/platforms/nec/.
11. SEMATECH[2] The national technology roadmap for semiconductors, 1997 edition, 1997. Ordering and downloading from URL: http://www.sematech.org/public/roadmap/index.htm.
12. S. Turek. Some software and algorithmic concepts of feast, 1998. To be publsihed in this proceddings.

[2] SEMATECH and the SEMATECH logo are registered servicemarks of SEMATECH, Inc.

Subject Index