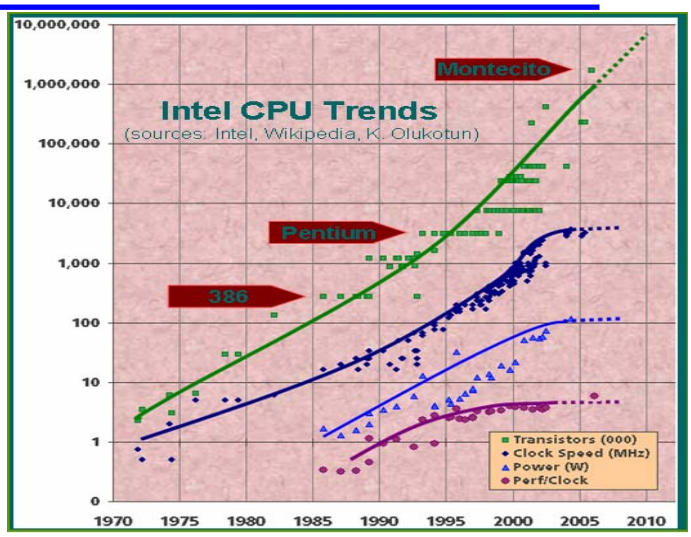
Reinventing High Performance Computing

Burton Smith
Technical Fellow

Microsoft

Times are Changing



[Thanks to Herb Sutter]

Microsoft

Parallel Computing is Now Mainstream

- Single processor performance is leveling off
  - Instruction-level parallelism is near its limit (the ILP Wall)
  - Power per chip is getting painfully high (the Power Wall)
  - Caches show diminishing returns (the Memory Wall)
- Meanwhile, logic cost ($ per gate-Hz) continues to fall
  - How are we going to use all that hardware?
- We expect new "killer apps" will need more performance
  - Semantic analysis and query
  - Improved human-computer interfaces (*e.g.* speech, vision)
  - Games!
- Microprocessors are now multi-core and/or multithreaded
  - But so far, it's just "more of the same" architecturally
  - How are we going to program such systems?

Microsoft

The ILP Wall

- There have been two popular approaches to ILP:
  - Vector instructions, including SSE and the like
  - The HPS† canon: out-of-order issue, in-order retirement, register renaming, branch prediction, speculation, …
- Neither scheme generates much concurrency given a lot of:
  - Control-dependent computation
  - Data-dependent memory addressing (*e.g.* pointer-chasing)
- In practice, we are limited to a few instructions/clock
  - If you doubt this, ask your neighborhood computer architect
- Parallel computing is necessary for higher performance

† Y.N. Patt et al., "*Critical Issues Regarding HPS, a High Performance Microarchitecture*," Proc. 18th Ann. ACM/IEEE Int'l Symp. on Microarchitecture, 1985, pp. 109–116.

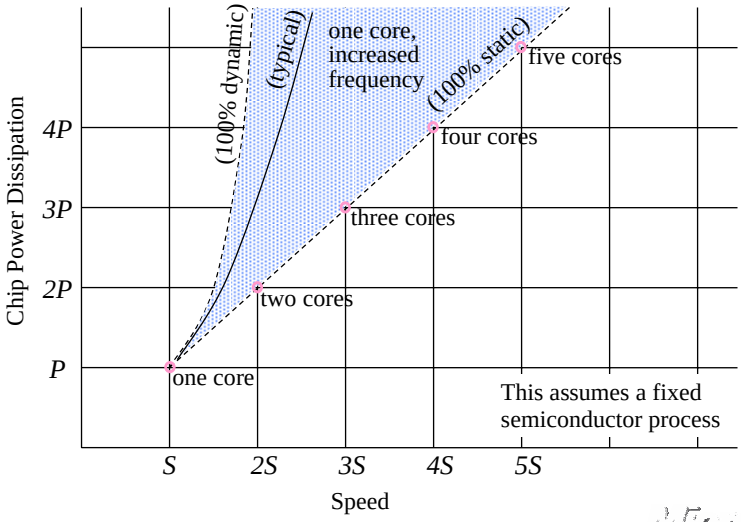
Microsoft

The Power Wall

- There are two ways to scale speed by a factor σ:
  - Scale the number of (running) cores by σ
    - Power will scale by the same factor σ
  - Scale the clock frequency *f* and voltage V by σ
    - Dynamic power will scale by σ^3 (CV^2f)
    - Static power will scale by σ ($Vi_{leakage}$)
    - Total power lies somewhere in between
- Clock scaling is worse when σ > 1
  - This is part of the reason times are changing!
- Clock scaling is better when σ < 1
  - Moral: if your multiprocessor is fully used but too hot, scale down voltage and frequency rather than processors
- Parallel computing is necessary to save power

Microsoft

Power vs. Speed



Microsoft

The Memory Wall

- We can build bigger caches from more plentiful transistors
  - Does this suffice, or is there a problem scaling up?
- To deliver twice the performance with the same aggregate DRAM bandwidth, the cache miss rate must be cut in half
- How much bigger does the cache have to be?†
  - For dense matrix-matrix multiply or dense LU, 4x bigger
  - For sorting or FFTs, the square of its former size
  - For sparse or dense matrix-vector multiply, forget it
- Faster clocks and deeper interconnect increase miss latency
  - Higher performance makes higher latency inevitable
- Latency and bandwidth are closely related

† H.T. Kung, "*Memory requirements for balanced computer architectures*," 13th International Symposium on Computer Architecture, 1986, pp. 49–54.

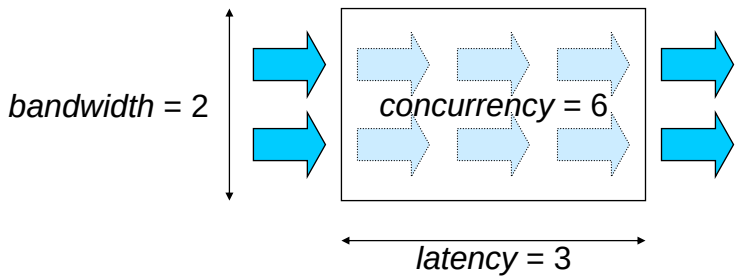
Microsoft

Latency, Bandwidth, & Concurrency

- In any system that transports items from input to output without creating or destroying them,

  latency x bandwidth = concurrency

- Queueing theory calls this result *Little's Law*



Microsoft

Overcoming the Memory Wall

- Provide more memory bandwidth
  - Increase aggregate DRAM bandwidth per gigabyte
  - Increase the bandwidth of the chip pins
- Use multithreaded cores to tolerate memory latency
  - When latency increases, just increase the number of threads
  - Significantly, this does not change the programming model
- Use caches to improve bandwidth as well as latency
  - Make it easier for compilers to optimize locality
  - Keep cache lines short
  - Avoid mis-speculation in all its forms
- Parallel computing is needed for processor/memory balance

Microsoft

The von Neumann Assumption

- Namely, “there is a single program counter”
  - Mainstream computing has relied on it for about 60 years
- Now this (and some things it brought along) must change
  - Serial execution lets programs *schedule values into variables*
  - Parallel execution makes this scheme hazardous
- Serial programming is easier than parallel programming
  - But serial programs are now becoming slow programs
- We need parallel programming paradigms that will make everyone who writes programs successful
- The stakes for our field’s vitality are high
- Mainstream computing must be reinvented

Microsoft

Consequences for HPC

- HPC has been a lonely parallel outpost in a serial world
  - Parallel computing is now becoming mainstream
- Consequences for HPC are likely to be:
  - A broadening spectrum of programming language choices
  - Routine combining of shared memory and message passing
  - Adaptation and use of mainstream software for HPC
- Successful HPC product offerings might include:
  - HPC editions of client applications and tools
  - HPC services that enable or accelerate client applications
  - HPC systems that scale up the client architectural model
- HPC will also be reinvented

Microsoft

Lessons From the Past

- A great deal is already known about parallel computing
  - Programming languages
  - Compiler optimization
  - Debugging and performance tuning
  - Operating systems
  - Architecture
- Most prior work was done with HPC in mind
  - Some ideas were more successful than others
  - Technical success doesn't always imply commercial success

Microsoft

Parallel Programming Languages

- There are (at least) two promising approaches:
  - Functional programming
  - Atomic memory transactions
- Neither is completely satisfactory by itself
  - Functional programs don't allow mutable state
  - Transactional programs implement dependence awkwardly
- Data base applications show the synergy of the two ideas
  - SQL is a "mostly functional" language
  - Transactions allow updates with atomicity and isolation
- Many people think functional languages are inefficient
  - Sisal and NESL are excellent counterexamples
  - Both competed strongly with Fortran on Cray systems
- Others believe the same is true of memory transactions
  - This remains to be seen; we have only begun to optimize

Microsoft

Shared Memory with Message Passing

- This topic has been a tough challenge for HPC
  - Some "give up" and deploy an MPI process per PC
- OpenMP is not very well-suited to message passing
  - The fork-join SPMD nature of the language is one problem
- Instead, we need languages that can do both:
  - Nested parallelism on local memory-resident data structures
  - Parallel message sends and receives to other address spaces
- This is absolutely necessary for many-core client systems
  - Local parallelism plus access to web data and services
- It will also make multi-core HPC nodes more productive

Microsoft

Compiler Optimizations for Parallelism

- Some say automatic parallelization is a demonstrated failure
  - Vectorizing and parallelizing compilers (especially for the right architecture) have been a tremendous success
  - They have enabled machine-independent languages
  - What they do can be termed *parallelism packaging*
  - Even manifestly parallel programs need it
- What failed is *parallelism discovery*, especially in-the-large
  - Dependence analysis is chiefly a local success
- *Locality discovery* in-the-large has also been a non-starter
  - Locality analysis is another word for dependence analysis
- The jury is still out on large-scale *locality packaging*
- In any event, the mainstream needs optimizing compilers
  - This will benefit our HPC customers as well

Microsoft

Parallel Debugging and Tuning

- Today, debugging relies on single-stepping and **`printf()`**
  - Single-stepping a parallel program is seldom effective
- Conditional breakpoints have proven to be valuable
  - For both program and data
- Support for ad-hoc data perusal is also very important
  - This is a kind of data mining application
- Serial program tuning tries to discover where the program counter spends most of its time by sampling it
- In contrast, parallel program tuning tries to discover places where there is insufficient parallelism
  - A proven approach has been event logging with timestamps
- We will want to extend these tools for our HPC customers
  - To achieve a single integrated view of the application
  - To get a higher level, more scalable user interface

Microsoft

Operating Systems for Parallelism

- Operating systems must stop trying to schedule processors
  - Their job should be allocating processors and other resources
  - Resource changes should be negotiated with the user runtime
- Work should be scheduled at user level
  - There's no need for a change of privilege
  - Locality can be better preserved
  - Optimization becomes much more possible
  - Blocked computations can become first-class
- Quality of service is important for many mainstream uses
  - Deadlines are more relevant than priorities in such cases
- Demand paging is a bad idea for most parallel applications
  - Everything ends up waiting on the faulting computation
- Windows will steadily improve for parallel clients and HPC

Microsoft

Parallel Architecture

- Hardware has had a head start at parallelism
  - That doesn't mean it's way ahead!
- Artifacts of the von Neumann assumption abound
  - Interrupts, for example
  - Most of these are pretty easy to repair
- A bigger issue is support for fine-grain parallelism
  - Thread granularity depends on the amount of state per thread and on how much it costs to swap it when the thread blocks
- Another is whether all processors should look the same
  - There are good reasons for heterogeneity
    - Heterogeneous architectures or heterogeneous implementations
  - Shared memory can be used to communicate among them
  - Homogeneous architectural data types will help performance
- The biggest issue may be how to maintain system balance

Microsoft

Conclusions

- We are now rethinking many of the basics of computing
- There is lots of work for everyone to do
  - I've left some subjects out, especially applications
- HPC has given us valuable experience with parallelism
  - Much of it will be applicable going forward
- HPC will benefit from mainstream parallel computing

Microsoft