

KILLER MICROS II: THE SOFTWARE STRIKES BACK

**BURTON SMITH
TECHNICAL FELLOW
EXTREME COMPUTING GROUP
MICROSOFT RESEARCH**

HPC Software is Primitive

- HPC programming is too low-level
 - C++, OpenMP, and MPI
- HPC tools are few and thinly supported
 - The market just isn't big enough
- HPC applications are too narrowly applicable
 - Assume ample statically mapped data parallelism
- HPC operating systems are stripped down
 - Resource management, communications, services

Changes Are Afoot

- The many-core inflection
 - Many processor cores in each socket
 - Parallel computing is becoming mainstream
- Heterogeneous processors
 - Able to do more than just graphics
 - High performance and low power consumption
- Cloud computing
 - Better data search and access
 - Service-based application software

Client Parallelism

- Client computing is becoming parallel
 - Even on mobile devices
- There are two reasons:
 - Continued performance improvement
 - Reasonable power/energy efficiency
- Parallel languages and tools are emerging
 - They are needed to use the new hardware well

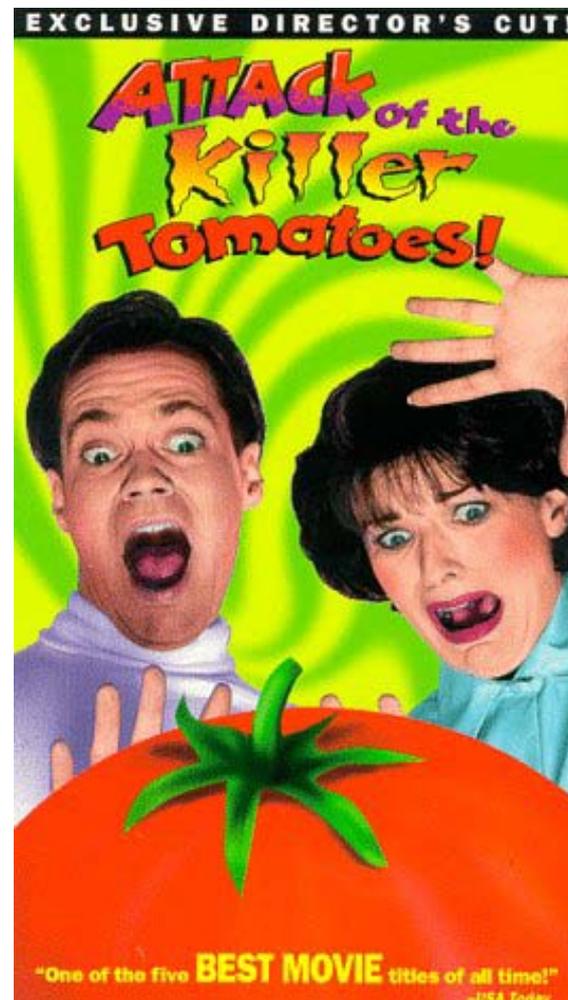
The Attack of the Killer Micros

At Supercomputing '89, Eugene Brooks predicted mainstream hardware would dominate the HPC arena:



“No one will survive the attack of the killer micros!”

Now it's the software's turn.



Some Consequences

Traditional Cluster Software

- SPMD (OpenMP) on nodes
- Static node and core counts
- Static load balancing
- MPI
- C++, Fortran
- File system storage

Emerging Mainstream Software

- Mixed parallelism on nodes
- Adaptive node and core counts
- Dynamic load balancing
- MPI, WCF, HTTP, XML...
- C++, F#, Excel, PLINQ, Dryad...
- File systems, databases, clouds

Mainstream software will have far richer capabilities

HPC software will derive from the mainstream

Shared Memory vs. Message Passing

- The two concepts are complementary
 - Shared memory enables concurrent access to state, whereas message passing is functional
- It is important they work very well together
- Unfortunately, in traditional HPC they don't
 - MPI calls are relegated to serial code regions
 - OpenMP is the main culprit
- PPL, TPL and Intel's TBB are much better
 - General task graphs with dynamic task scheduling

Dynamic Task Scheduling

- The pros:
 - Automatically balances the computing load
 - Lets software use any number of cores
 - Helps tame asymmetric cores
- The cons:
 - Overhead
 - Progress guarantees (*e.g.* priority inversions)
- *User-Mode Scheduling* addresses the cons

User-Mode Scheduling

- Basic idea: don't use the OS to block/unblock
- It has a long history, usually not well-reported
 - *E.g.* <http://en.wikipedia.org/wiki/Thread>
- Win7's UMS + Dev10's ConcrRT implements it
- Why block/unblock in user mode?
 - To make it very inexpensive
 - To let applications manage affinity
 - To provide full observability of state
 - To enable more sophisticated synchronization

Non-Blocking

- Non-blocking synchronization was invented to improve parallel computing on kernel threads
 - *E.g.* to pull the fangs of priority inversions
- The bad news:
 - Mutual exclusion becomes intrinsically optimistic
 - Complex state mutation becomes even more complex
 - Message passing and I/O become asynchronous operations from the programmer's point of view
 - Exception handling is still more daunting to “get right”
- Both control flow and data flow become tangled

Data-Parallel Computation

Application	SQL	Sawzall	≈SQL	LINQ, SQL
Language		Sawzall	Pig, Hive	DryadLINQ Scope
Execution	Parallel Databases	Map- Reduce	Hadoop	Dryad Cosmos, HPC, Azure
Storage		GFS BigTable	HDFS S3	Cosmos Azure SQL Server

<http://connect.microsoft.com/Dryad>

Heterogeneous Computing

- Our GPU partners all want similar things:
 - Ability to use GPUs in multiple roles concurrently
 - Shared virtual memory among CPUs and GPUs
 - Fast synchronization among CPUs and GPUs
- Shared memory is great for synchronization
 - User mode blocking/unblocking can schedule the unblocked continuations to the appropriate ISA
 - I/O, *e.g.* messaging, can also be handled this way
 - Interrupts can be limited to “dumb” devices

Resource Management

- User-mode scheduling allocates cores (or HW threads), not kernel threads, to processes
 - This makes cores much more like other resources: memory, various bandwidths, *etc.*
 - Processes ask for (and sometimes yield) resources
 - The OS arbitrates among the competing processes
- A key question: what is the policy machinery?
 - I think I have a handle on this for the client case
 - It will vary between clients and servers, *e.g.* HPC
 - One common theme is the need to provision SLAs

Service-Level Agreements

- In clients, some applications need a minimum resource quota from the OS to work properly
 - They are called Quality Of Service (QOS) apps
 - QOS apps have a Service Level Agreement (SLA) that the app must meet, *e.g.* output per frame at some frame rate, to keep the app's user happy
- SLAs are standard fixtures in cloud computing
 - Communicating services need to have an SLA
- I believe a solution for the SLA provisioning problem is in reach, even for client systems

Service Oriented Architecture

- We want applications composed from services
 - So services can be near(er) data, cycles, or energy
 - So services can be relocated as needed
- This concept has interesting implications
 - Clients and servers should have the same APIs
 - Inter-process message passing should be unified
 - All services ask for resources to support their SLAs
- Clients and servers start to look a lot alike
- That's why MS Technical Computing has both

Non-Blocking Is Non-Necessary

- User mode scheduling makes blocking work
 - High contention on mutual exclusion isn't fatal
 - Complex state mutations don't have to distill everything into a single hardware operation
 - Message passing and I/O stay synchronous to the programmer (but asynchronous in the runtime)
 - Exception handling can await outcomes, whether from message receipt or state modification
- Its result is more productive programming
- It also helps clean up a few other problems

SOA Inside HPC Systems

- The idea is to compose applications as parallel workflows within HPC systems
 - Distributed file systems resemble this vision
 - HPC applications become service pipelines
 - Data locality is improved over storage-based flows
- Techniques used for client OS can help
 - Load balancing is a crucial function
 - If services can run anywhere, just balance SLAs

HaaS: HPC as a Service

- SOA allows seamless use of HPC as a Service
 - For smallish problems, the app runs on a client
 - For big ones, a part of the app stays home and the rest moves to an HPC system
- HaaS in the cloud would be useful
 - Dedicated corporate HPC might be unnecessary
 - Even HPC in home clouds might be worthwhile

Conclusions

- HPC software is pretty primitive, chiefly because the market is small
- Major computing revolutions now underway will change the HPC landscape
- Mainstream software will take over HPC
- Client, HPC and Cloud software will converge
- Technical Computing will benefit from all this