Petascale and Parallel Programming:

Everything you learned in kindergarten is wrong

Joshua Fryman, Ph.D.
Senior Research Scientist
Microprocessor Technology Labs
Intel Corporate Technology Group
joshua.b.fryman@intel.com

© 2007 Intel Corporation
Example: Server CPU Tick-Tock Model

8-Jul-08 Talk at CScADS Workshop

All dates, product descriptions, availability and plans are forecasts and subject to change without notice.

What about Larrabee?
The New Future

• A plethora of transistors . . .
  – But scalar performance glory is essentially over
  – Off-chip bandwidth has always been critical
  – Limits to gains through cache alone

• Many-core days are upon us
  – Multi is a small numer (1, 2, 4, 8 . . .)
  – Many is not a small number

• How do we utilize many-core well?
  – Start over with everything?!
Slightly Heterogeneous Designs?

• Architectural knobs:
  • How many big cores? Their hardware threads?
  • How many little cores? Their hardware threads?
  • What’s the interconnect between group(s)?
Alternate Future Platform?

General Purpose Cores
Special Purpose HW
Interconnect Fabric

Major Heterogeneous Multi-Core
If you build it . . .

• They will not know how to use it
• Anyone can get some gain from many-core
  – Few can get good gain from it
• ISA evolution complicates scaling
  – Fractures, fixed function units, OoO and InO
• Where did we go wrong?
  – EECS 101?
  – Kindergarten?
EECS 101 == Kindergarten?

• Write down a (serial) list of things to do
• Share Everything
• Communicate, talk about what you’re doing
• Don’t push others
• Hold hands, stick together
• Take a nap every day (during lecture)

• We have a problem here . . .
• We (all) trained the world to be: serial
  – Ooops, now what?
“There must be some way out of here”

• Tuning and Opportunities:
  – Re-think all EECS curriculum points?
  – Self-tuning floating point hardware?
  – Break the ISA contract?
  – Hide latency with massive hardware threading?
  – Extended languages with “new” tools?
  – Develop (yet another) new parallel language?
Self-tuning Floating Point H/W?

- I want float-64 ... -128 ... -256
  - Really only need float32 – mostly!
- Instability when RD/U/N/Z mismatch
  - Why not compute all at once?
  - Throw a flag when tolerance exceeded
  - ISA-level support for best performance
- Need more semantic knowledge to use
  - Do we really need higher precision?
  - Or is programmer being lazy?
- More than just per-instruction issue
  - Needs software support to handle bodies
Breaking the ISA contract?

• How does an FMA get used?
  – Unsafe compiler optimization, or
  – Programmer hand-coding

• Why does FMA get used?
  – IEEE 0.5ulp result?
  – Or just because it’s faster than Mul-Add?

• Need more semantic information
  – **Why** is this (operation) here?
  – Need to capture high-level programmer intent
  – This intent can express more ideas for additional gains
    – Associativity, Vertical Mining, Horizontal Utilization
• Both approaches use hw threads to get more done
• CPUs have low thread count, expect complex work
• GPUs have high thread count, expect simple/similar work
• How to *divide* the work is a semantic knowledge problem
• There may not *exist* a single point of convergence
• But this is today’s problem . . .
Extensions: Data Parallel Operations

Data parallelism provides a rich set of types & operations

- Element-wise operations
- Reductions
- Nested parallelism
  - trees, graphs, sparse matrices, …
- Arbitrary communication

- Allows for better scaling for uArch issues (SIMD width, cores)
- Side-effect free in native Data Parallel constructs
- Can be mixed with legacy apps/code via extensions (Ct)
- Hard to dynamically tune for without semantic keys
Putting it Together: Ct

• [www.intel.com/go/ct](http://www.intel.com/go/ct)
• Explicit Data-Parallel Constructs in C/C++
• Support for abstract data structures
  – Sparse, Dense, Nested, Flat, . . .
• Automatic thread creation for TLP as well
  – Balances between MIMD and SIMD (CPU/GPU)
  – Adjusts for varying uArch parameters
• Lazy binding and aggressive JIT
  – Live optimization better than profiled/static
Conclusion: The Semantic Gap

- Programmers taught to baby-step problems
  - Works, but destroys key *semantics*
  - Ex: How to tune for SSE, AVX, LRB, GPU, and future?
  - Many different techniques needed to unroll/schedule

- We need to capture better semantics in binary
  - Develop a standard for cross-platform use

- Then ISA, auto-tuning/re-optimizing, BT use it!
  - Obtain better performance than otherwise possible
  - Optimally generate code for . . .
  - Massive MIMD, massive SIMD, or any point in between

- Semantics *hard* but can no longer ignore