Parallel Languages: Past, Present and Future

Katherine Yelick

U.C. Berkeley and Lawrence Berkeley National Lab
Internal Outline

- Two components: control and data (communication/sharing)
- One key question: how much to virtualize, i.e., hide machine?
  - Tradeoff: hiding improves programmability (productivity), portability, while exposing gives programmers control to improve performance
- Important of machine trends
  - Future partitioned vs. cc shared
  - Transactions will save us
- PGAS: what is it? What about OpenMP?
  - Looking ahead towards multicore: these are not SMPs. Partitioned vs cc shared memory
- What works for performance: nothing virtualized *at runtime*, except Charm++
- Open problem: load balancing with locality
Two Parallel Language Questions

• What is the parallel control model?
  - data parallel (single thread of control)
  - dynamic threads
  - single program multiple data (SPMD)

• What is the model for sharing/communication?
  - implied synchronization for message passing, not shared memory
  - shared memory
  - message passing
Petaflop Desktop By 2026?

1 PFlop system (100K cores?)

- SUM
- #1
- #500
- Desktop

Expon. (Desktop)

6-8 years

8-10 years

Slide source Horst Simon, LBNL
HPC Programming: Where are We?

- BG/L at LLNL has 64K processor cores
  - There were 68K transistors in the MC68000
- A BG/Q system with 1.5M processors may have more processors than there are logic gates per processor
  - Trend towards simpler cores, but more of them

- HPC Applications developers write programs that are as complex as describing where every single bit must move between the transistors in the MC68000
- We need to *at least* get to “assembly language” level

Slide source: Horst Simon and John Shalf, LBNL/NERSC
A Brief History of Languages

• When vector machines were king
  • Parallel “languages” were loop annotations (IVDEP)
  • Performance was fragile, but there was good user support

• When SIMD machines were king
  • Data parallel languages popular and successful (CMF, *Lisp, C*, …)
  • Quite powerful: can handle irregular data (sparse mat-vec multiply)
  • Irregular computation is less clear (multi-physics, adaptive meshes, backtracking search, sparse matrix factorization)

• When shared memory multiprocessors (SMPs) were king
  • Shared memory models, e.g., OpenMP, Posix Threads, are popular

• When clusters took over
  • Message Passing (MPI) became dominant

We are at the mercy of hardware, but we’ll take the blame.
Partitioned Global Address Space Languages

- **Global address space:** any thread may directly read/write data allocated by another \(\rightarrow\) shared memory semantics
- **Partitioned:** data is designated local/remote \(\rightarrow\) message passing performance model

- **3 older languages:** UPC, CAF, and Titanium
  - All three use an SPMD execution model
  - Success: in current NSF PetaApps RFP, procurements, etc
  - Why: **Portable** (multiple compilers, including source-to-source); **Simple** compiler / runtime; **Performance** sometimes better than MPI
- **3 newer HPCS languages:** X10, Fortress, and Chapel
  - All three use a dynamic parallelism model with data parallel constructs

Challenge: improvement over past models that are just large enough
Open Problems

• Can load balance if we don’t care about locality (Cilk)
  • Can we mix in locality?
  • If user places the work explicitly can we move it? They can unknowingly overload resources at the “place” because of an execution schedule chosen by the runtime

• Can generate SPMD from data parallel (ZPL, NESL, HPF)
  • But those performance results depend on pinning
  • E.g., compiled a program and run it on P processors, what happens if task needs to use some of them?

• Can multicore support better programming models?
  • A multicore chip is not an SMP (and certainly not a cluster)
    • 10-100x higher bandwidth on chip
    • 10-100x lower latency on chip
  • Are transactions a panacea?
Predictions

• Parallelism will explode
  • Number of cores will double every 12-24 months
  • Petaflop (million processor) machines will be common in HPC by 2015 (all top 500 machines will have this)

• Performance will become a software problem
  • Parallelism and locality are key will be concerns for many programmers – not just an HPC problem

• A new programming model will emerge for multicore programming
  • Can one language cover laptop to top500 space?

• Locality will continue to be important
  • On-chip to off-chip as well as node to node