The “State” and “Future” of Middleware for HPEC

Anthony Skjellum
RunTime Computing Solutions, LLC
&
University of Alabama at Birmingham

HPEC 2009

Overview

- Context
- Tradeoffs
- Standards
- Futures
- Summary

COTS Computing -> COTS APIs and Middleware

- With COTS computing, the combined goals for
  - Performance
  - Portability
  - Productivity
- And lack of vendor “lock” has motivated use of High Performance Middleware

High Performance Computing Meets Software Engineering

- Kinds of software maintenance
  - Perfective
  - Adaptive
  - Corrective
- Standards allow for adaptive maintenance, including porting to new platforms
- Standards “allow” vendors to deliver performance and productivity in narrow areas without “undoing” COTS; compete within confines of interfaces
- Standards remove need for each development project to make a layer of adaptive middleware
- Porting to new platforms important in COTS world

Tradeoffs

- Layers can’t add performance, unless they raise the level of specification and granularity of operations and allow compile-time and/or runtime-optimization
- More lines of standard code vs. concise unportable fast vendor specs?
- Small reductions of achievable performance can yield huge in portability and productivity
- There is no guarantee that users use standards or vendor-specific codes optimally

Tradeoffs

- Middleware standards are specified as if they are libraries and API calls
- Middleware standards are implemented as libraries with API calls
- Middleware standards could be implemented as domain-specific languages and/or compiler-based notations, with lower cost of portability (but implementations more expensive)

Parallel Compilers for HPEC

- There is no effective automatic parallel compilation… no change in sight
- OpenMP directives helpful but not a general solution for thread parallelization (e.g., in GCC 4.x)
- High Performance Middleware still has to “pull its weight”

Standards that emerged in HPEC

- Drawing on HPC
  - MPI
  - MPI/RT (with DARPA support)
- Unique to HPEC
  - VSIPL
  - DRI
  - VSIPL++
- CORBA

Two classes of adoption

- Significant
  - VSIPL
  - MPI
  - VSIPL++ (which also leverages DRI)
- Not significant
  - MPI/RT
  - DRI (although ideas used elsewhere)
- CORBA successfully used, but not for HPC purposes usually, rather distributed computing; so important, but “adjacent”

Why MPI is successful

- Successful in HPC space
  - revolutionized software development in DOD/DOE/academia
  - Even limited commercial application space
- Widely implemented
  - free versions
  - Commercial productions
- Widely available training in colleges
- Easy to write a program in MPI
- Matches the CSP model of embedded multicomputers
- Developers generally use MPI 1.x subset only in HPEC space (standardized in 1994)

Why VSIPL Successful

- Close mapping to “FPS library” notation from the past
- Removes vendor lock from math functions (e.g., non-portable functions)
- Eliminates “accidental complexity” in apps
- Math functions are hard to implement on modern processors
- ADT abstractions (object based) useful
- Very good performance in practice
- Not hard to use

Why VSIPL++ is Successful

- For those that use C++
  - More performance possible
  - Less lines of code possible
- DRI concepts included
- Parallel abstraction included
- Acceptance of the parallel model, C++, and template meta programming is a big leap for some developers/programs, but is worthwhile

DRI - Still Important

- DARPA Data Reorganization Initiative
- Works to replace 1 function family in MPI - Alltoall
- Key HPEC abstraction (Corner Turn)
- All the best ideas for Corner Turn in the field put in the standard
- Created at end of “DARPA Push” in the area… no customer demand; picked up in VSIPL++
- Best ideas already in vendor libraries like PAS
- Next steps: Try to get into MPI-3 standard

MPI/RT Not Successful

- Easy to implement without QoS
- QoS needed OS support (lacking)
- Hard to write programs in MPI/RT
- Addressed performance-portability issues in MPI-1.1 area, but not compelling enough
- Implementations done on CSPI, Mercury, Sky?, Radstone, Linux Cluster… but did not reach critical mass of customer demand/acceptance
- MPI/RT Concepts remain important for future middleware standardization
- Huge effort invested, no disruptive change in development

I. Performance-Productivity-Portability Space

- Goal is to tradeoff
- You can’t have it all
  - Choose at Most 1
  - Choose at most 2
- Those tradeoffs are different as you move to different COTS architectures
  - Tuning always needed
- Performance-Portability in Middleware helps
- Productivity Matters too, but often sacrificed to get the other two factors

Price of Portability

- Due to Richard Games (Ken Cain involved too)
- Classical example:
  - MPI_Alltoall* function
  - Much slower than hand coded
  - Just proves that MPI specified this function about as suboptimally as it could have
  - Side effect: motivated DRI
- Better examples:
  - MPI Send/Receive vs. low level vendor DMA
  - VSIPL vs. chained optimized pulse compression in vendor-specific math API

Price of Productivity

- Productivity doesn’t have to lower Performance (e.g., meta programming)
- But it often does
- Productivity enhancing interfaces in HPEC… the key one is VSIPL++

Price of Resilience (A 4th Dimension)

- All standards discussed so far lack a fault tolerant model
- Space-based and even terrestrial embedded multicomputers have fault scenarios
- No significant effort to refactor successful APIs done yet
- HPEC approaches to fault awareness needed
- Aspect-oriented type approaches possible

II. Investment in Further Standardization

- There is very limited current investment in these areas
- MPI-3 - not working on performance issues still bogging down MPI-1.x functions used in HPEC [NSF funding MPI-3 meeting attendance]
- VSIPL - dormant; latest standard efforts were not key to addressing more performance or more productivity
- No significant investments from DARPA now… DARPA and/or NSF need to drive
- Actually, there was always limited investment :-) it is just more limited now

III. Broader Adoption

- Must be driven by ultimate customers (e.g., USN)… if not required by customer, why will primes or other developers use standards?
- Legacy codes that mix standards and vendor APIs are still vendor locked
- Economic models that show value must be enhanced
- Standards must be enhanced for newer, more complex heterogeneous architectures
- Standards must be implemented well on target platforms [vendors fully embrace]

IV. Newer Architectures

- Heterogeneous
- Multithreaded / multicore
- GPU + CPU
- CPU + FPGA
- Balance changes in communication, computation, and I/O
- Need for better overlap
- All mean that the programming models of existing standards have to be augmented or revamped
- Commercial standards like OpenCL and CUDA… we have to interoperate with these

Summary

- Overall, middleware, and principally MPI and VSIPL, have driven up the capability of defense and other HPEC applications to be performance-oriented and portable at the same time. Significant legacy applications have been developed,
- These standards have proven useful.
- Others as well (e.g., POSIX)

Summary

- How do we get more benefit from middleware?
- Where do we invest, what private and government stakeholders make those investments, when, and how?
- Investment long overdue and can be extremely beneficial, and even enhance the competitiveness of HPEC systems vendors.
- Leverage from HPC?
- Does a lean budget drive more interest or less?