Toward the Embedded Petascale Architecture: The Dual Roles of Middleware and Software Engineering in Future HPEC

Anthony Skjellum, PhD

Department of Computer and Information Sciences
http://www.cis.uab.edu

September 22, 2011

Background

- Community has worked on standards in HPC and HPEC for the last 18 years (MPI, VSIPL, DRI, MPI/RT, …)
- Evolution of thought and practice on abstraction and encapsulation and OO in Signal/Image Processing for HPEC
- Evolution of HPEC systems over time : Open, Linux, POSIX, common APIs - VSIPL, MPI
- Systems of 1-4 Single precision Gigaflops in 1999 … Goal of PetaFlop in 2018
- Benefits from open source (e.g., GCC, Linux)

Exascale requires unprecedented integration of concerns

- Apps
- OS
- Runtime
- System
- 3rd party codes
- Compilers
- Fault models
- Optimizers/Schedulers
- Global constraints, utility functions, schedules, QoS

Exascale is Unforgiving

- Global state is not scalable
- Device granularities vary
- Many kinds of heterogeneity
- “Where” and “when” to compute and store??
- Systems will evolve quickly
- “Tweaking” and “Tuning” current apps insufficient

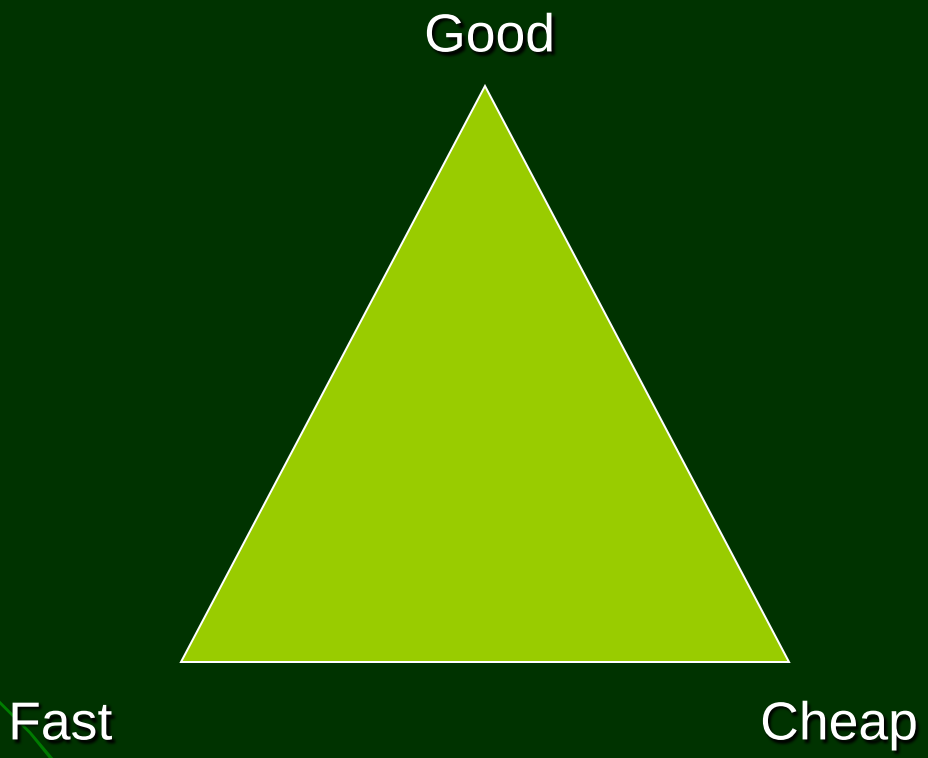
Embedded Petascale

- Shares some concerns of Exascale, even though 1/1024 scale
- Size-weight-power issues may be even more demanding compared to HPC
- Additional concerns for embedded deployable and rugged
- Exascale Systems composed of Embedded Petascale Systems?
- Reuse of Exascale technologies and vice versa appears crucial

High level view of hardware

- We know that hardware will be massively parallel (millions of entities)
- Heterogeneous in memory and compute units
- Subject to many faults
- Power, space, location, and other constraints apply
- We know there will be multiple approaches and varied instantiations
- Embedded Petascale Systems will be scale models of Exascale Systems

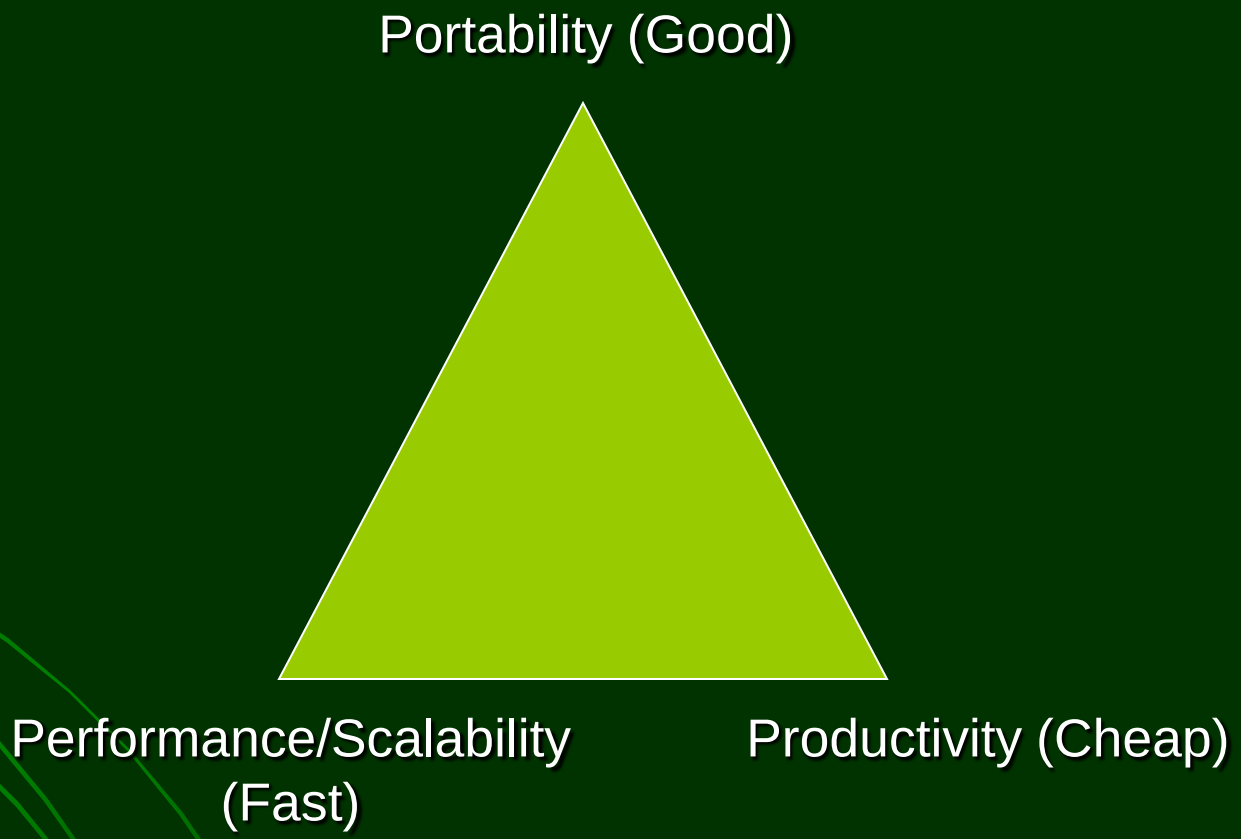
Engineering Maxim: Good-Fast-Cheap - choose 2



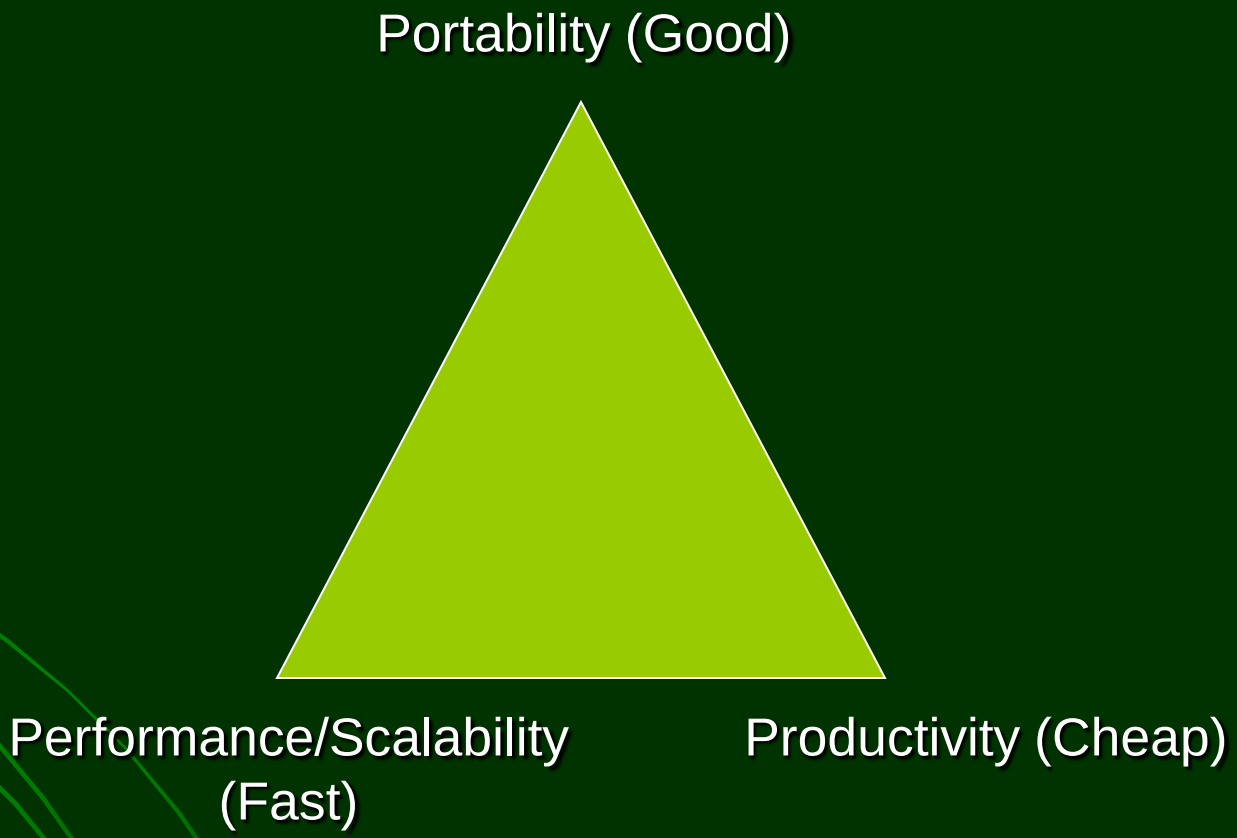
http://en.wikipedia.org/wiki/Project_triangle

Wexelblat's Scheduling Algorithm: Choose two: Good; Fast; Cheap.

Parallel Computing Maxim: Choose at Most 2



Exascale Computing Triangle: Choose at Most 1?



Baseline Assumptions

- Combinatorial optimization and parameter exploration is needed to achieve high performance
- Polyalgorithm libraries are crucial to achieve high performance (providing optimizers choices with common contracts)
- For many years, it has been clear that there are many algorithms for the same problem (at fixed or varying accuracy) … e.g., dense matrix multiplication has huge algorithmic space
- Managing complexity for the application is needed

Advanced Software Engineering Needed

- Every generation of HPC has a big compiler-does-it-all push
- Every generation of HPC has “we do it all by hand” push
- Systems initially hard to program, and those successful highly rewarded with performance
- Applications for defense worth many billions to keep running overshadow cost of systems and TCO for some sectors… new systems constraints to run “legacy” apps
- Good enough results needed before systems obsolesce

Advanced Software Engineering Important

- Embedded Petascale, like Exascale, has unprecedented complexity
- Managing millions of resources difficult
- Software has to be reusable to be affordable
- Fault tolerance and QoS are needed
- Large teams of developers will be used

MOPA - Advanced Software Engineering

- Models (MDA) - Managing Lifecycle and Structure and Complexity
- Objects - Managing Complexity
- Patterns - Reusable Experience
- Aspects - Global Constraints

Model Driven Architecture

- Develop all software as large scale Software for Embedded Petascale
- Use models to Support “Large Scale Construction” and enable complete life cycle Management of applications
- Treat many more classes of “Code” as “write only”
- Support plug-ins for optimization, compilers, third-party components

Object-orientation

- This is not just using C++, also UML, …
- Object-oriented everywhere as a principle for all artifacts underlies modern software engineering
- For embedded petascale, OO means that algorithms are abstracted, contracts are abstracted, and there is an ability to mix, match, and switch implementations
- Middleware represented here as standard families of objects, hierarchies, and contracts

Patterns

- Design patterns (cf, Erich Gamma and many others) are crucial to large-scale software
- Design P\patterns for Embedded Exascale together with Families of Components that can describe, implement the patterns will be helpful
- Lots of research lately on “parallel computing patterns”… right direction
- Patterns are an engineering way to reuse “tribal knowledge”

Aspects (AOP)

- AOP is the “responsible way” to apply global constraints to systems
  - Fault models
  - Energy monitoring
  - Cross abstraction barriers (runtime, app, middleware, scheduler)
- Supports separation of concerns in new dimensions, and also compartmentalization
- Can be generalized (e.g., Semantic Designs products, Rose Framework) to be extensible compiler technology… languages become extensible

We need a lot of reusable, efficient, modeled software

- The kind of middleware we need for embedded petascale
  - Relevant to domains for HPEC
  - Validated / Trusted / Modeled
  - Work within a lattice
  - Support free and commercial
  - All competing and collaborating entities
  - Support “product lines”
- Need common, community scoped repositories (app stores)

What becomes of VSIPL, MPI

- The best reusable, indivisible operations of communication and computation go into component libraries
- Patterns of communication and computation are captured
- VSIPlets and MPIlets (and shmem or PGAS) reside in component libraries (together with characteristics and implementations)
- Everyone no longer writes their own FFT, and FFTs come with a finite number of flexible contracts

MOPA+Compilers put it all together

- While component libraries can be assembled by hand, working with models and aspects…
- Compilers have to take the framework of assembled applications, and choose subsets of components, do offline and online optimization
- Compilers compile, Optimizers Optimize, Tuners Tune
  - We need to make sure as much is open and can be substituted as possible… monoliths a bad idea

MOPA+Compilers put it all together

- The application always is evolving in the face of changing constraints: either to new modes, or to new hardware, or to new levels of QoS
- So a compilation of software is a set of options to run at runtime, and MDA provides means for validation and cataloging of all those “production lines”
- (We learned a lot before and during DARPA PCA)

Combinatorial Optimization Crucial

- Offline optimization good for “predefined” modes of operation (cf, DARPA PCA)
- Online optimization needed for changing conditions (e.g., faults)
- Software engineering validation of “predefined” modes crucial for HPEC deployments
- Ability to run at risk, rather than stop working completely will be of value in some fault scenarios
- This is also called “autotuning”
- Compilers, schedules and programs use “combinatorial” optimization, not just compilers

How to build community of MOPA for Petascale HPEC?

- Need an investor (e.g., DARPA)… or investors
- Need long term transition plan so it doesn't "go away"
- Need the community to support the diversity of stakeholders and contributors
- Participation must be economically viable (competition, cooperation, hybrid, government contributors, freeware)
- Validation, Classification, Lattices, and Trust all crucial

How to build community of MOPA for Petascale HPEC?

- Scoped for HPEC communit(ies) so it can be used with confidence
- Ability to include artifacts that “make us stronger” and reject the rest
- No one will invest in apps until critical mass present
- We then need emerging compiler developers to interface with the components
- Can’t standardize until there is maturity (!?!?)
- Need to ensure that key architectures supported
- Many parts should be open to universities, but some parts proprietary and etc

Summary

- Exascale = Embedded Petascale * 1024
- Open Systems Crucial
- Runtime abstractions must be minimal expensive
- Software Engineering Advances Must be Exploited:
  - Objects, Models, Aspects, Patterns
- Integration, Reuse of Components and Models Crucial
- Need communities of reusable, validated, deployable components for Exascale (commercial and free)
- We can’t keep reinventing the wheel, the software will be too expensive
- Combinatorial and Compiler Optimizations Key
- Middleware (Standards Based) Useful - Framework for Component Libraries

UAB COLLEGE OF ARTS AND SCIENCES

Questions? Discussion?

Contact info: *tony@cis.uab.edu*
http://www.cis.uab.edu