Hardware Breakout Group

What can hardware provide to software and applications

- Abstract machine model
- Goal start as general as possible and drill in only as needed
- Levels of information
  - Could be specifications at different level
    - Ex: cache characteristics
  - Could be spreadsheet model
    - Scaling trends, how do things scale
    - Sensitivity to different design parameters
  - Could be simulators
    - As above information becomes more complex

Providing continued…

- System models from previous slide should include tradeoffs of bottlenecks
  - Methods needed to express the tradeoff
- The ability to model tradeoffs of bottlenecks will allow a chance of being able to balance across all the applications
  - 7 codesign centers, 7applications each, IESP, ESC, etc. etc etc
  - Need a structure to the interaction
- Categories of things to tradeoff
  - $$$
  - Power
  - Bandwidth: memory, network, I/O
  - Flop/ratios
- A description of how the tradeoffs interrelate
  - When a particular feature goes up what feature(s) go down, sensitivity
- Need information to flow across the stack, so they do not conflict

Input Feedback – Setting the Context

- All features of hardware are important to applications
  - When asked do you want more bandwidth, less latency, etc
  - Answer is yes
- Architecture involves making tradeoffs, cost versus schedule versus power versus bandwidth versus latency versus …
- Key to architecture tradeoffs is finding what is the bottleneck for an application allowing better understanding how a particular tradeoff will affect an application
  - Cost of bottlenecks – how much $, how severe
  - How strict is bottleneck, how easy/hard to trade off against other
  - Timing of bottleneck, if we wait a little does this bottleneck go away
- System models from previous slide should include tradeoffs of bottlenecks
  - Methods needed to express the tradeoff
- Can application questions we incorporated into the application model

Application Information Useful to Architectural Decisions (Highlights)

- Love hearing about science, but we need to translate the exascale challenges that have been identified to machine characteristics
- Hardware/System software group need to talk the right language

- How much of your application is amenable to running on cores that are largely computation
- How much of your application will meaningful benefit from branch prediction, Out of Order, etc

- How well can you predict memory access pattern
- How much memory does your node need
  - Minimum requirements
  - Threshold points
- How is the memory used
  - What is it used for: big memory table, read only, how widely accessed, how much redundant state (we need to work on together with tools)
- For a 1TF node what point does memory bandwidth become the bottleneck
- How sensitive to NUMA

Application Information Useful to Architectural Decisions (Highlights)

- How important is resilience to your application
  - Need definition of resilience
- There is a recognition the I/O and storage issue are becoming more important
  - Need to better understand whether this is because “that’s the way it is done” or are they inherent
- Minimum synchronization timing of the code
  - What types (allreduce, barrier, pt-to-pt), frequency,
- How many threads per MPI task
  - Would you want
  - Could you utilize
- Hierarchy of parallelism
  - Need a sense of how
- How would you utilize an exascale machine
  - Job mix, capability versus capacity, etc.
- Need to have an application model
  - Spreadsheet, analytical, executable specific and application wide
  - Need to have criteria on applications for producing this information

Information Useful to Architectural Decisions (full set)

- Memory
  - More or less memory needed per process? Value of non-coherent shared memory?
  - What is the MB/flop trend for main memory?
  - What is the MB/MPI process trend for main memory? Are there enough interesting applications where this number is below 512 MB?
  - What is the MB/thread trend for main memory? (Gives an idea about the number of threads that will be needed.)
- RAS Expectations
  - How is the MTBF required of applications going to change? (Are there going to be longer jobs using larger number of processors?)
  - How willing will users be to (a) manage recovery after failure detected by hardware, and (b) manage both detection and recovery by appropriate duplication of code?
  - How fine grain of check pointing would be considered acceptable assuming we can achieve >90% availability?
  - Willingness to run with degraded performance for some amount of time? How long?
- Job Mixes
  - What type of job mix (sizes) do you expect? This is needed to help us understand the importance of things like full system bisection versus something like 1/4 system bisection. The net is different topologies have different scaling of local communication to bisection and as usual there is no free lunch. Need to give up on some of one to get more of the other
  - Will different applications be required to run on the same processor? (Implications for protecting resources)
- Scaling:
  - Expectation for mostly weakly scaled?
  - Perceived scaling limitations now and in future workloads? (load balance, strong scaling limits, messaging scaling problems,)

Information Useful to Architectural Decisions (full set)

- Programming Style
  - How is SIMD adoption changing in your applications? (Implications on SIMD width)
  - Is there any significant movement from AOS to SOA? (Implications on cache and bandwidth utilization)
  - Is there any movement away from sparse matrix algorithms? (Need to be competitive with GPUs)
  - Can applications tolerate unordered reductions? (Makes hardware simpler)
  - Will low TLB activity continue to be the norm? (Implications on efficiency of handling page faults)
  - Is there move towards greater use of libraries? (Implications on optimization)
- Will applications mostly be legacy or new in 2015? 2018?
  - What will be the pre-dominant programming model (s) -
  - Continued evolution of C/C++ and Fortran with MPI/OpenMP?
  - Any use of scripting languages (Python, Ruby, Perl etc)?
  - Any PGAS languages?
  - How much interest in new programming languages? How much focus should we put on optimization of the system for new languages versus current languages?
- Performance
  - Views on using profiling techniques to improve performance
  - Any thoughts on dynamic optimization/recompilation for performance and/or porting legacy code?
  - Any general existing collection of actual application performance data would be very interesting.

- How do you expect the applications to change in the 2015/2018 time frame? In particular... more or less compute as compared to communication ?
- Perceived limitations to threading of MPI processes. difficulty eliminating the serial portion? difficulty with thread scaling due to... thread false cache line sharing? duplicated work? overheads to resume and join? Loops not deep enough?
- Degree of threading expected to be commonplace by 2015
- How persistent are the send/receive patterns and collectives from time step to time step. (We could imagine getting very good hardware/software acceleration/optimization if the persistent case is the common case. This is persistent in terms of actually doing the same thing not in terms of using MPI persistent syntax).