BSC vision Towards Exascale

Jesús Labarta, Eduard Ayguade and Mateo Valero,
Barcelona Supercomputing Center – Centro Nacional de Supercomputación
(BSC–CNS) and Universitat Politècnica de Catalunya
Nexus II Building, C. Jordi Girona 29, 08034–Barcelona, Spain.
{ jesus.labarta, eduard.ayguade, mateo.valero}@bsc.es.

What is scalability? We need to reach a view of our systems, where looking at them from different distances still lets us have a self similar view, like looking at the earth form the moon, a satellite, a plane, the top a mountain or standing on the ground. We need unified views of our computing systems, where **granularity** is the main difference between the levels we may focus at.

The current experience represents a single snapshot of different techniques at various granularity levels. We use dataflow ideas in out of order processor design, decoupling between logical and physical address space at the virtual memory level, synchronous algorithms at cluster level, … . We should look at all good ideas, developments and practices form the past and apply them in a broad scalable way, where granularity is the only difference between levels.

Power, variance, resource (energy budget, processing power, storage, and communication) sharing and management, and global complexity are important challenges. Memory structure is a key issue, seen both form the point of view of the model offered to the programmer and the actual hardware structure and support mechanisms. Overlap between communication and computation and in general better tolerance to latency is important to avoid the over dimensioning of communication infrastructures that current practice seems to favor.

Facing these challenges, **asynchronism and decoupling** different conceptual levels is the key to tackle a universe huge in scale and characterized by variance. We consider that the programming model is Alexander's sword to break the Gordian knot of multicore and exascale systems. A proper **programming model** is the key interface that will allow the separation at a coarse granularity level between the concerns of users and those of system designers in the same way ISAs did allow such separation and progress in the past. The only issue is that we still have for forge this sword and we will require strong interaction of all levels to do it.

We need programming models that help decouple the way programs are written and executed. At the programmer interface, we should be able to write ideas left to right, top to bottom in a clean and concise way. Runtime should be able to execute them out of order (right to left, top to bottom,…) in the way that the utilization of the resources and thus global efficiency is optimized.

Ideally a single programming model with hierarchical capabilities should cover the whole dynamic range from the single node to the exascale system. It is nevertheless foreseeable that mixed approaches will be used in the near future, with a different model being used for the cluster, node and accelerator/device level. In this situation, an issue that will have strong impact on the programmability of our systems and the final performance is the "compatibility" between the models at the different levels. Different models have/promote different parallelization and synchronization structures. Very often in the past, mixed models gave discouraging results due to a mismatch between the parallelization structure at the coarse and fine grain level. Considering that properly parallelizing an application (and we are really facing Amdahl's law) is a global issue, both programming model designers and application writers need to put special care in ensuring that the interactions between the fine and coarse level result in positive interference.

We consider that a clean specification of what are the inputs/outputs/accesses of a computational block (**task model**) is a proper boundary between a programmer who has a good knowledge of the algorithmic interactions and the execution engine. The runtime should be responsible of the scheduling issues: progressing as fast as possible along the critical path; knowing which functional units (cores) are more appropriate for each task; deciding where to issue task to maximize locality and minimize bandwidth requirements. Mechanisms for the programmer to provide hints and additional information to the runtime will be useful, but not a requirement.

We need to **decouple memory** as a logical address space to name objects from memory as container to keep the values. Matching objects to the actual containers available should be handled dynamically by the run time, in a much more flexible than what is today done. We are used to a single level of such mapping being handled by hardware (caches,…) and we will need to consider a hierarchical approach, where at coarse levels of granularity this functionality is handled by the runtime.

Many of these ideas come from dataflow, yes, and they should be extensively used in our execution engines. We do need syntactical ways to provide a **smooth transition** form current practices to facilitate the adoption of such techniques by the huge community of programmers who are scientist, but not computer scientist.

At the application level we do **need to restructure our codes** to clearly reflect the actual access patterns. Many current applications have accesses to key global structures deeply buried into the call tree. This is not only bad for the future exascaling of the code, it is also bad for today's maintenance and development of new functionalities. We envisage that such application cleaning process will have to be undertaken by application developers in their way to exascaling. This will have to be done while including in the code the asynchrony and means to determine dependences between computations. What would be important is to ensure that this is an **only once effort**, leading to applications that can survive for some decades and can be upgraded and rapidly ported to the foreseeable explosion of hardware platforms. This should be feasible in a **portable,**

modular and incremental way, possibly tuning some low level task description to specific accelerator hardware but leaving the program structure and code unmodified.

At the application level it will also be important to work on new **algorithms that are more asynchronous in nature**. It is hard to imagine programs with tens of millions of threads synchronizing globally at fine granularity that will run efficiently and insensitive to variance or noise. It will be necessary to study where the balance stands in terms of computational complexity of an algorithm and the level of asynchronism that it has.

Load balancing is a key issue to achieve performance at high scale, which is frequently underestimated. We tend to believe that our applications are more balanced than they really are if their actual execution is measured in fine detail. Very often in current practice we blame the communication subsystem when the real cause of the problems comes from load imbalances or serializations. MPI, like a perfect gas, fills whatever space you give it. We should look more at what and how we compute and a bit less at how much time we spend in MPI. **Dynamic load balancing** techniques will have to be used to solve the issue, irrespective of whether it is caused by the application itself, or originates from variance in the devices or system software or from the shared usage of resources. In the same way that having to continuously use force to enforce power is not having real power these dynamic techniques should be always there, but only enforced when needed.

Malleablility of applications is a feature that will be a requirement mostly arising as a requirement of shared utilization of systems and the attempts to optimize the global throughput of systems and quality of service/SLAs. Malleability, as the ability of an application to change its parallel structure (change resources used) is a feature that will have to be enabled/facilitated by programming models, although application developers will have to follow a few methodological guidelines. The same techniques developed for load balancing above will be needed in the runtime to achieve malleability. The only difference is that in this case, the decisions will have to be coordinated with the OS schedulers at the different levels (kernel threads, processes, jobs).

Fault Tolerance will certainly be a relevant issue as it will not be possible to ensure functional operations of all the components of a system for the execution time of applications. The failures may happen at different granularity levels (individual functional units or cores, whole address spaces…) Techniques to tolerate these faults will be needed. Depending on the granularity may be implemented in software or hardware support may be required. Task based programming models as advocated above in conjunction with transactional memory functionalities seem to provide a fair basis to approach the issue. Faults if properly handled, recovered and isolated will result in dynamic availability of resources, thus linking back with the load balance issue described above.

Understanding the performance of our programs will be of great importance. We have the feeling that performance tools and analysis practices are a bit in their infancy. Today we essentially measure some aspects of system performance and report very global aggregates that generally convey little information about the details, and unfortunately, it is in the details where a lot of the performance of these systems will be gained or lost. There is a need (and potential) for much more statistical processing of our data, use of analysis techniques from other areas (i.e. signal processing, clustering) more extensive use of models in order to actually provide insight to the analyst.

At BSC we have been working on the StarSs programming model, which we believe addresses in a clean way many of the above stated considerations. Initial implementations of the run time for SMP, Cell, GPUs are available. It can be integrated with other models at large cluster scale (i.e. MPI) and still propagate to such an outer level many of the benefits of the dataflow execution. Also further implementations of the basic model at coarser granularity levels are being explored. We do believe that ideas from the model can on one side guide and on the other highly benefit form architectural support, especially in the memory subsystem design area. We are also involved in performance tools, job scheduling, applications… We would like to contribute with our vision and ongoing efforts to this holistic Exascale initiative.