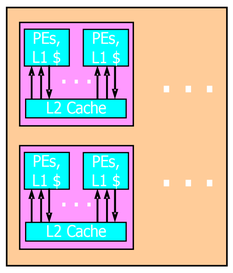
IBM X10 Team

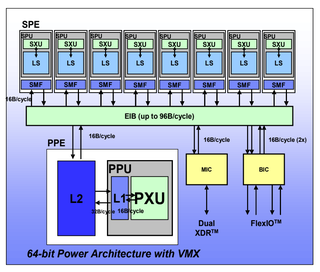
The Challenge

Parallelism scaling replaces frequency scaling as foundation for increased performance ⇒ Profound impact on future software

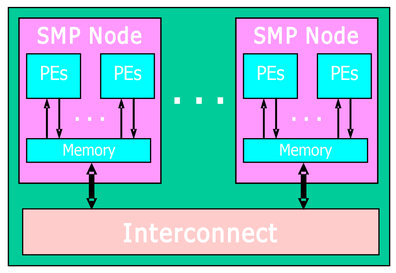
Multi-core chips

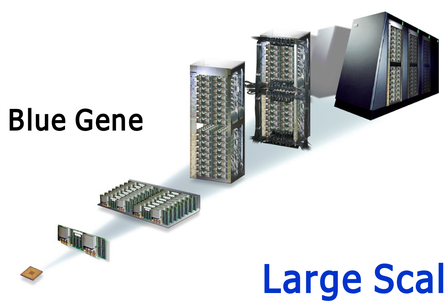


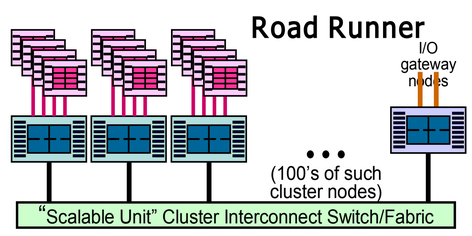
Heterogeneous Parallelism



Cluster Parallelism





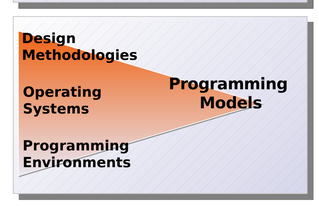


Large Scale Parallelism

Programming Models: Bridging the Gap Between Programmer and Hardware

- A programming model provides an abstraction of the architecture that enables programmers to express their solutions in manner relevant to their domain
  - Mathematicians write equations
  - MBAs write business logic
- Compilers, language runtimes, libraries, and operating systems *implement* the programming model, bridging the gap to the hardware.
- Development and performance tools provide the surrounding *ecosystem* for a programming model and its implementation.

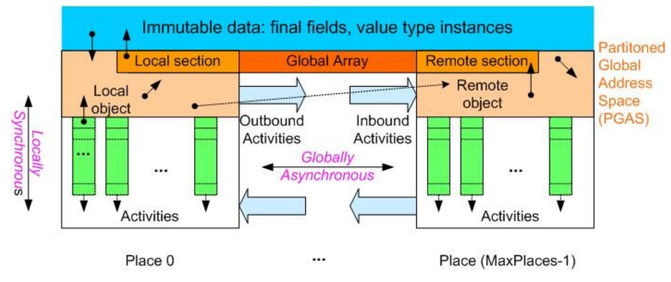
- The evolution of programming models impacts
  - Design methodologies
  - Operating systems
  - Programming environments



APGAS Realization

- Through languages
  - Asynchronous Co-Array Fortran
    - extension of CAF with asyncs
  - Asynchronous UPC (AUPC)
    - Proper extension of UPC with asyncs
  - X10 (already asynchronous)
    - Extension of sequential Java
- Language runtimes share common APGAS runtime through an APGAS library in C, Fortran, Java
  - Implements PGAS
    - Remote references
    - Global data-structures
  - Implements inter-place messaging
    - Optimizes inlineable asyncs
  - Implements global and/or collective operations
  - Implements intra-place concurrency
    - Atomic operations
    - Algorithmic scheduler
- Libraries reduce cost of adoption, languages offer enhanced productivity benefits

Asynchronous PGAS Programming Model



Fine grained concurrency • async S	**Atomicity** • atomic S • when (c) S	**Global data-structures** • points, regions, distributions, arrays
Place-shifting operations • at (P) S	**Ordering** • finish S • clock	

Two basic ideas: Places and Asynchrony

APGAS Advantages

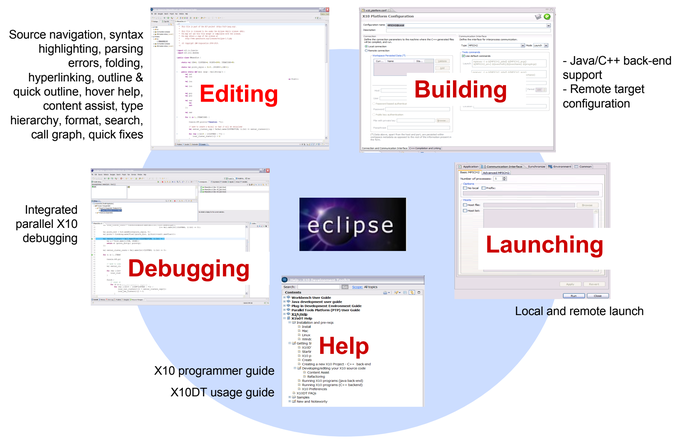
- Programming model is still based on shared memory.
  - Familiar to many programmers.
- Place hierarchies provide a way to deal with heterogeneity.
  - Async data transfers between places are not an ad-hoc artifact of the Cell/GPU
- Asyncs offer an elegant framework subsuming multi-core parallelism and messaging.
- There are many opportunities for compiler optimizations
  - E.g. communication aggregation.
  - So the programmer can write more abstractly and still get good performance
- There are many opportunities for static checking for concurrency/distribution design errors.
- Programming model is implementable on a variety of hardware architectures
  - Leads to better application portability..
  - There are many opportunities for hardware optimizations based on APGAS

X10 Project Status

- X10 is an APGAS language in the Java family of languages
- X10 is an open source project (Eclipse Public License)
  - Documentation, releases, implementation source code, benchmarks, etc. all publicly available at http://x10-lang.org
- X10 and X10DT 2.1 released mid-October 2010
  - Simplified distributed object model
  - Enhanced language support for common async/finish idioms
  - Significant improvements in X10DT functionality and robustness
  - Support for executing X10 code on cluster of GPGPUs:
    - Annotated async bodies automatically compiled to execute on GPU via CUDA.
- X10 2.1 Platforms
  - Java-backend (compile X10 to Java)
    - Runs on any Java 5 JVM
    - Single process implementation (all places in one JVM)
    - Multi-process implementation in progress
  - C++ backend (compile X10 to C++)
    - AIX, Linux, cygwin, MacOS, Solaris
    - PowerPC, x86, x86_64, sparc, BG/P
    - Multi-process implementation (one place per process)

X10 DT: Integrated Development Tools for the X10 Programmer

A modern development environment for productive parallel programming



X10 Results on UTS

- Simplified kernel for global load balancing
- Problem: Count number of nodes in a dynamically generated, unbalanced tree.
  - Each node described completely by 32 bytes – so computation is completely relocatable.
  - High compute/spawn ratio
- Client of X10 Global Load Balancing Framework

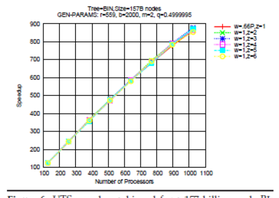


Figure 6. UTS speedup achieved for a 157 billion node BINOMIAL tree on IBM's P7HV32 cluster. The speedup shown is based on the sequential performance of UTS of the same problem, which is 2.7 million nodes per second. The parameter w indicates the number of random steals attempted before resorting to lifelines. The parameter z indicates the number of lifelines for each compute node. The UTS revision number used was 15294.

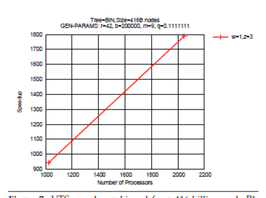


Figure 7. UTS speedup achieved for a 416 billion node BINOMIAL tree on Argonne National Lab's Blue Gene/P cluster. The speedup shown is based on the sequential performance of UTS, which is 0.54 million nodes per second. The parameter w indicates the number of random steals attempted before resorting to lifelines. The parameter z indicates the number of lifelines for each compute node. The UTS revision number used was 15277.

X10: Productivity and Performance at Scale