Investigation of Programming Models for Emerging FPGA-Based High Performance Computing Systems

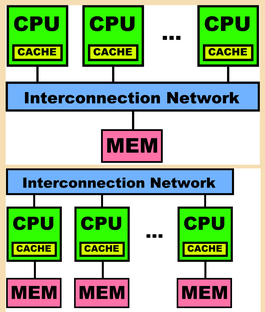
Andrew W. H. House and Paul Chow
{ahouse, pc}@eecg.toronto.edu
Department of Electrical and Computer Engineering
University of Toronto, Toronto, ON, Canada M5S 3G4

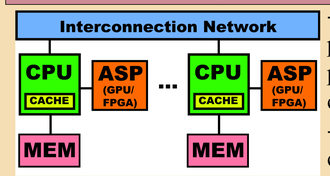
The Landscape of High Performance Computing

Recent Trends:
- Microprocessor performance gain curve is flattening.
- Demand for high performance computing (HPC) continues to grow.
- Application-specific processors (ASPs) are now a viable alternative.

Result:
- Landscape of HPC is undergoing rapid change.
- HPC systems using ASPs (FPGAs, GPUs, Cell processors) are emerging.
- We can classify these systems as follows:

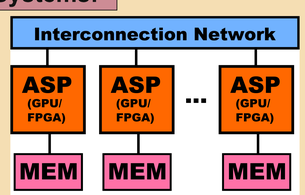
Multiprocessor Systems:
- Multiple general-purpose microprocessors.
- Shared-memory systems.
- Distributed-memory systems.
- Examples: IBM BlueGene, Beowulf clusters, SETI@home

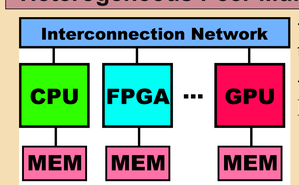


Accelerated Multiprocessor Systems:


- ASPs are subordinate to a general-purpose microprocessor to accelerate a particular aspect of the overall computation.
- Examples: Cray XD-1, SGI RASC, GPU computing

Application-Specific Multiprocessor Systems:
- ASPs are directly connected to create a high performance computing system.
- Removes CPU as bottleneck.
- Examples: multi-FPGA systems (BEE2, TMD)



Heterogeneous Peer Multiprocessor Systems:


- CPUs and ASPs are peers with equal access to system resources.
- Examples: Folding@home, Cray XT5h, DRC/XtremeData

Problem is in Programming:
- Different processing elements have unique programming challenges.
- Each class of system provides its own programming challenges.
- Systems must be easy to use and optimize the use of each kind of computational resource.

Programming Model Requirements

What does the programming model need to support?
- Emerging architectures.
- Programming HPC applications.
- User productivity.

From this, we propose a set of requirements, and consider the implications of each.

Requirements

1. The programming model must support heterogeneous processing elements.
- SPMD models no longer work.

2. The programming model must be scalable.
- Explicit communication and distribution of data is problematic.

3. The programming model must avoid features that are difficult to synthesize.
- Pointers and recursion should be avoided.

4. The programming model should assume only limited system services.
- No dynamic memory allocation.
- Only basic system I/O and computation supported.
- “Embedded Supercomputing”

5. The programming model should support many types and classes of computation.
- Provide rich set of data types.
- Provide support for different computational “Dwarfs”.

6. The programming model should allow the programmer to expose as much fine- and coarse-grain parallelism as possible.
- Bit-, word-, data- and task-parallelism.
- Implicit parallelism where possible.

7. The programming model should enforce separation of algorithm from implementation.
- Optimization handled at back-end.

8. The programming model must be independent of the architecture it runs on.
- Programs must be portable between machines.
- Model cannot assume any particular architecture.

9. The programming model should provide a degree of execution model transparency.
- Programmer must understand what will cause communication / computation without having the details of how.

Evaluating Existing Programming Approaches

Symbol	Meaning
×	Language does not support or prevents meeting the requirement.
✓	Language supports or meets the requirement.
	Language has no particular support, but does not prevent.
✓× or ×✓	Language can support one requirement when another is unsupported, and vice versa.

Model	Language	Heterogeneous Processing	Scalable	Synthesizable	Limited System Resources	Many Types of Computation	Expose parallelism	Separate algorithm and implementation	Architecture independent	Execution model transparency
Data Parallel	HPF	×	✓		×	×	✓	×	×	✓
	C*	×	✓	×	×	×		×	✓	×
	Dataparallel C	×	✓	×	×	×		×	×	×
	mpC	✓	×	×	×	×		×	×	
	RapidMind	✓		×	×	×		×	✓	
Dataflow and Functional	Simulink	✓	×		✓		✓	×	×	✓
	LabVIEW	✓	×		✓		✓	×	×	✓
	Prograph	✓	×		✓		✓	×	×	✓
	Lustre	✓	×		✓		✓	×	×	✓
	Multilisp		×		×	×	✓		✓	×
	SISAL	×			✓	×	✓	×	✓	
	SAC	×				×	✓	×		
	Cilk	×		×		×	✓	×		×
	CellSs	✓	×	×		×		×	×	×
	Mitrion-C			✓	✓	✓	✓	×	×	×
Stream Computing	StreamIT	×	×		✓	×		×	✓	
CSP	Occam	✓	×		✓	×		×	✓	
	MPI	×✓	✓×		×	✓	×	×	×	
	PVM	×✓	✓×		×	✓	×	×	×	
	Active Messages	×	×		×	✓	×	×	×	
HDLs	Verilog/VHDL	✓	×	✓	✓	✓	✓	×	×	✓
	Handel-C	✓	×	✓	✓	✓	✓	×	×	✓
Shared Memory	PRAM	×			×	✓	×	×		
	SHMEM	×		×	×	✓	×	×	✓	
	OpenMP	×		×	×	✓	×	×	✓	
	Linda	✓	×	×	×	×	×	×	✓	
	Orca	×	×	×	×	×	✓	×	✓	
PGAS	Split-C	×	✓	×	×		×	×	×	
	UPC	×	✓	×	×	✓	×	×		
	Titanium	×	✓		×	✓	×	×	×	
	Co-Array Fortran	×	✓		✓	×		×		
	ZPL	×	✓			✓	✓	×		
	Fortress		✓	×	×	✓	✓		×	
	X10		✓	×	×	✓	✓	×	×	
	Chapel	×	✓	×		✓	✓	×		
Parallel Objects	CAL	✓	×		×	×	×	×	✓	

Positive Features
- Data-parallel operations.
- Futures.
- Implicit communication.
- Easy synchronization.
- Functions without side effects.
- Global view of memory.
- Data streams.
- Region-based array management.

Negative Features
- An SPMD programming style.
- Dynamic memory allocation and pointers.
- Explicit introduction of all parallelism.
- Explicit communication between specific processes.
- Explicit mapping of computation to processing elements.
- Only one level of parallelism, or one style of computation.
- Complex or costly process synchronization.
- Trade-off between scalability and heterogeneity.

Conclusions

- The emerging landscape of HPC requires new programming models.
- Programming models must satisfy a number of constraints.
- A set of requirements has been proposed as an evaluative tool.
- Existing programming models fail to satisfy all of the requirements.
- Analysis identifies language features that support/undermine requirements.