Distributed Shared Memory for High-Performance Computing

Stéphane Zuckerman Haitao Wei Guang R. Gao

Computer Architecture & Parallel Systems Laboratory
Electrical & Computer Engineering Dept.
University of Delaware
140 Evans Hall Newark,DE 19716, USA
{szuckerm, hwei, ggao}@udel.edu

Tuesday, October 17, 2014

Outline

1. **Introduction**
2. **Hardware Distributed Shared Memory: NUMA Systems**
   - Uniform Memory Access Systems
   - Non-Uniform Memory Access Systems
3. **Software Distributed Memory Systems**
   - Introduction to Global Partition Address Space Systems
4. **Where to Learn More**

Introduction I

Why Distributed Shared Memory?

- To ease the programmer's task $\Rightarrow$ productivity
- …And that is mostly it, really.

Why Is Productivity Important?

Let's ask Fred Brooks (Brooks, *The Mythical Man-month (Anniversary Ed.)* Chap. 8):

> *IBM OS/360 experience, while not available in the detail of Harr's data, confirms it. Productivities in range of 600–800 debugged instructions per man-year were experienced by control program groups. Productivities in the 2000–3000 debugged instructions per man-year were achieved by language translator groups. These include planning done by the group, coding component test, system test, and some support activities (…)*
> *Both Harr's data and OS/360 data are for assembly language programming. Little data seem to have been published on system programming productivity using higher-level languages. Corbatò of MIT's Project MAC reports, however, a mean productivity of 1200 lines of debugged PL/I statements per man-year on the MULTICS system (between 1 and 2 million words). (…)*

Why Is Productivity Important? (Cont'd)

Brooks, *The Mythical Man-month (Anniversary Ed.)* Chap. 8:

> *This number is very exciting. Like the other projects, MULTICS includes control programs and language translators. Like the others, it is producing a system programming product, tested and documented. The data seem to be comparable in terms of kind of effort included. And the productivity number is a good average between the control program and translator productivities of other projects. But Corbatò's number is lines per man-year, not words! Each statement in his system corresponds to about three to five words of handwritten code! This suggests two important conclusions.*
>
> - *Productivity seems constant in terms of elementary statements, a conclusion that is reasonable in terms of the thought a statement requires and the errors it may include.*
> - *Programming productivity may be increased as much as five times when a suitable high-level language is used*

Why Is Productivity Important? (cont'd)

Brooks, *The Mythical Man-month (Anniversary Ed.)* Chap. 12:

> *The chief reasons for using a high-level language are productivity and debugging speed (. . .) There is not a lot of numerical evidence [in the 1960s. . .], but what there is suggests improvement by integral factors, not just incremental percentages. (. . .) For me, these productivity and debugging reasons are overwhelming. I cannot easily conceive of a programming system I would build in assembly language.*

So really, productivity solves two major problems: Time-to-solution (*i.e.*, software is produced faster), and how to produce bug-free code

Introduction IV

Different Ways of Implementing DSMs

- Hardware
- Software
- Hardware-software hybrids

Uniform Memory Access

- SMP systems used to propose a uniform access to memory banks

  ⇒ Example: for x86, a single front-side bus (FSB) to access DRAM
- Advantages:
  - For the hardware, easier to design and implement
  - For the programmer, guarantees on latency
- Drawbacks:
  - To guarantee uniform access, throughput is somewhat slowed down
  - In general, UMA architectures do not scale beyond a single compute node.
  - Even on a single compute node, UMA systems saturate easily

Non-Uniform Memory Access Systems

- Hardware can be designed so memory banks are directly attached to a given (set of) socket(s)
- To maintain a single address space, an interconnection system must be implemented
  - In theory NUMA systems need not be coherent
  - In practice all NUMA systems currently available are really Cache Coherent NUMA (ccNUMA)
- Examples: x86-based multi-processor compute nodes provide an interconnection network:
  - AMD Opteron-based systems use HyperTransport
  - Intel Xeon-based systems use QuickPath Interconnect (QPI)
  - SGI proposed the Altix multiprocessor NUMA system (based on Intel Itanium2 processors) where an unmodified Linux OS could access up to 1024 processors (so up to 2048 cores)

Limits of (cc-)NUMA

- Even in the case of large-scale NUMA like Altix systems, scalability remains an issue:
  - At the hardware level: producing hardware for large-scale ccNUMA requires it to be tightly coupled with the processors
  - At the software level: ensuring data locality becomes a bigger problem
- At the operating system level: a choice must be made (by the user):
  - Let the OS follow a "first-touch" page allocation policy ⇒ best for when the software can easily be optimized for locality
  - Require the OS to allocate pages in a random or round-robin way (when data access is truly random-ish).
- For very large scale computations, an additional software layer must be implemented to help access the fast network devices (*e.g.*, Infiniband, Quadrics, *etc.*).

Basic Concepts

- Maintain a programmer-centric global address space
- "Automagically" partition arrays and other shared data structures across compute nodes
- Provide means to handle locality: if an object is supposed to be available locally, there should be a way to inform the system
- When shared data structures are accessed, the software automatically knows where to issue the request

How to Implement PGAS

- Using a library (*e.g.*, Gasnet)
- Using a programming language (*e.g.*, X10, Chapel, Titanium, ...)

A Very Brief History

DARPA's High Productivity Computing Systems (HPCS) program was launched in 2002 with five teams, each led by a hardware vendor: Cray Inc., Hewlett-Packard, IBM, SGI, and Sun Microsystems.

Examples of PGAS Languages

Several languages are following a PGAS approach: IBM's X10, Cray's Chapel, HP and Berkeley's Unified Parallel C (UPC), *etc.*
They all propose constructs to express parallelism in a more or less implicit way. Most of these languages are either developed as an Open Source package or propose an open implementation[a]

[a]This is important: languages get popular thanks to their availability or because they are the only ones on their "market segments!"

Chapel

- Official web site: `http://chapel.cray.com`
- Open Source (`https://github.com/chapel-lang`)
- Targets "general parallelism" (*i.e.* any algorithm should be expressible as a Chapel program)
- Separates parallelism and locality: concurrent regions of code vs. data placement.
- Multi-resolution parallelism: either use implicit parallelism, or if parallel expert, use direct parallel constructs to drive parallel execution
- Targets productivity (type inference, iterator functions, OOP, various array types)
- Data-centric

Task Parallelism Constructs

- `begin{...}`: Creates an anonymous task using the code between braces
- `cobegin{...}`: Fine-grain way to task creation – Creates a task for each statement in the block

Data Parallelism Constructs

- `forall elem in Range do ...`: Creates a coarse-grain parallel loop – akin to OpenMP's `#pragma omp for`
- `coforall elem in Collection do ...`: Creates a fine-grain parallel loop – each iteration is a task

Synchronization

- `sync{statement;}`: Creates a synchronization point for all tasks created within a parallel region. Akin to a barrier (coarse-grain synchronization)
- `var variableName sync` *`type`*`;`: Creates a (set of) synchronization variable(s) which acts as a full/empty bit (set of) location(s).
- `var variableName atomic` *`type`*`;`: Creates a (set of) variable(s) that are accessed *atomically* (accesses are sequentially consistent).

Locality Constructs

- The `Locale` type: used to confine portions of computations and data to a specific part of the machine (typically a compute node).
- The `on` clauses: to make a statement execute a specific locale.

Locality and parallelism constructs can be combined, *e.g.*, `begin on Locale.left {...}`

X10

- Official web site: `http://x10-lang.org`
- Open Source (`http://sourceforge.net/projects/x10/files/x10dt/2.5.0/x10dt-2.5.0-linux.gtk.x86.zip/download`)
- Built on top of Java VM
- Partially inspired by Scala (a mostly functional, but multi-paradigm language based on the JVM)
- Provides a back-end to both Java and C++ code (source-to-source translation)
- Also distinguishes between parallelism and locality

X10 II

Parallel Constructs

- `async{...}`: Creates an anonymous task using the code between braces
- `finish{statement;}`: Creates a synchronization point for all `async` tasks created within a parallel region.
  - Can be combined: `finish async{...}`

Locality Constructs: Accessing Places

- `at`: Place shifting operation
- `when`: Concurrency control within a place
- `atomic`: Concurrency control within a place
- `GlobalRef[T]`: Distributed heap management
- `PlaceLocalHandle[T]`: Distributed heap management

X10 III

Combining Locality and Parallelism Constructs

- `at(p) function(...)`: Remote evaluation
- `at(p) async function(...)`: Active message
- `finish for (p in Places.places()) { at(p) async runEverywhere(...)}` : SPMD
- `at(ref) async atomic ref() += v`: Atomic remote update

X10 Examples
Hello World

```
import x10.io.Console;

class HelloWorld {
  public static def main(Rail[String]) {
      Console.OUT.println("Hello␣World!" );
  }
}
```

X10 Examples

Hello World

```
import x10.io.Console;

class HelloWorld {
  public static def main(Rail[String]) {
      Console.OUT.println("Hello World!" );
  }
}
```

```
import x10.io.Console;
class HelloWholeWorld {
  public static def main(args:Rail[String]):void {
    if (args.size < 1) {
      Console.OUT.println("Usage: HelloWholeWorld message");
      return;
    }

    finish for (p in Place.places()) {
      at (p) async Console.OUT.println(here+" says hello and "+args(0));
    }
    Console.OUT.println("Goodbye");
  }
}
```

X10 Examples

Fibonacci

```
import x10.io.Console;

public class Fibonacci {

  public static def fib(n:long) {
    if (n<2) return n;

    val f1:long;
    val f2:long;
    finish {
      async { f1 = fib(n-1); }
      async { f2 = fib(n-2); }
    }
    return f1 + f2;
  }

  public static def main(args:Rail[String]) {
    val n = (args.size > 0) ? Long.parse(args(0)) : 10;
    Console.OUT.println("Computing␣fib("+n+")");
    val f = fib(n);
    Console.OUT.println("fib("+n+")␣=␣"+f);
  }
}
```

Learning More About Multi-Threading and OpenMP

Internet Resources

- General PGAS web site: `http://pgas.org`
- Chapel: `http://chapel.cray.com`
- X10: `http://x10-lang.org`
- UPC: `http://upc-lang.org`, `http://upc.lbl.gov/` and `http://upc.gwu.edu`
- The GASNet library (used in Berkeley's UPC): `http://gasnet.lbl.gov/`

Tutorials Used for this Class

- Bradford L. Chamberlain's overview of Chapel: `http://chapel.cray.com/papers/BriefOverviewChapel.pdf`
- Chamberlain's slides to present Chapel: `http://chapel.cray.com/presentations/ChapelForETH-distributeme.pdf`
- X10 tutorial slides: `http://x10.sourceforge.net/tutorials/x10-2.4/APGASProgrammingInX10/APGASprogrammingInX10-slides-V7.pdf`
- UPC tutorial slides: `http://upc.gwu.edu/tutorials/tutorials_sc2003.pdf`

Food for Thoughts

- Sutter, "The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software" (available at `http://www.gotw.ca/publications/concurrency-ddj.htm`)
- Lee, "The Problem with Threads" (available at `http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-1.pdf`)

- Boehm, Hans-J. "Threads Cannot Be Implemented As a Library". In: *SIGPLAN Not.* 40.6 (June 2005), pp. 261–268. ISSN: 0362-1340. DOI: 10.1145/1064978.1065042. URL: http://doi.acm.org/10.1145/1064978.1065042.
- Brooks Jr., Frederick P. *The Mythical Man-month (Anniversary Ed.)* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN: 0-201-83595-9.
- Lee, Edward A. "The Problem with Threads". In: *Computer* 39.5 (May 2006), pp. 33–42. ISSN: 0018-9162. DOI: 10.1109/MC.2006.180. URL: http://dx.doi.org/10.1109/MC.2006.180.
- Sutter, Herb. "The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software". In: *Dr. Dobb's Journal* 30.3 (2005).