The Chapel Memory Consistency Model

Michael Ferguson
Sung-Eun Choi
Elliot Ronaghan
Greg Titus
Cray Inc.

June 13, 2015

Safe Harbor Statement

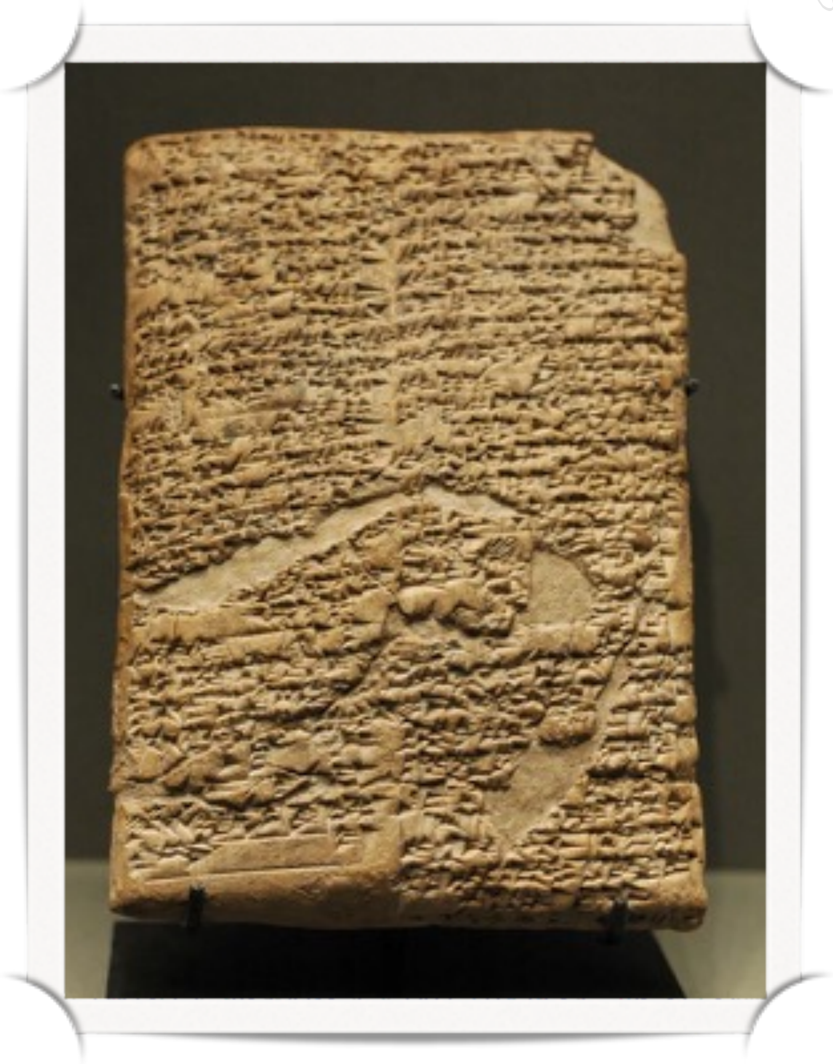
This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

The Memory Consistency Model Effort

- Philosophically:
  - The memory model already exists
  - We're just writing it down

Philosophy of the MCM Effort

- The memory model already exists
  - in example programs
  - in developer's minds
- We're just writing it down



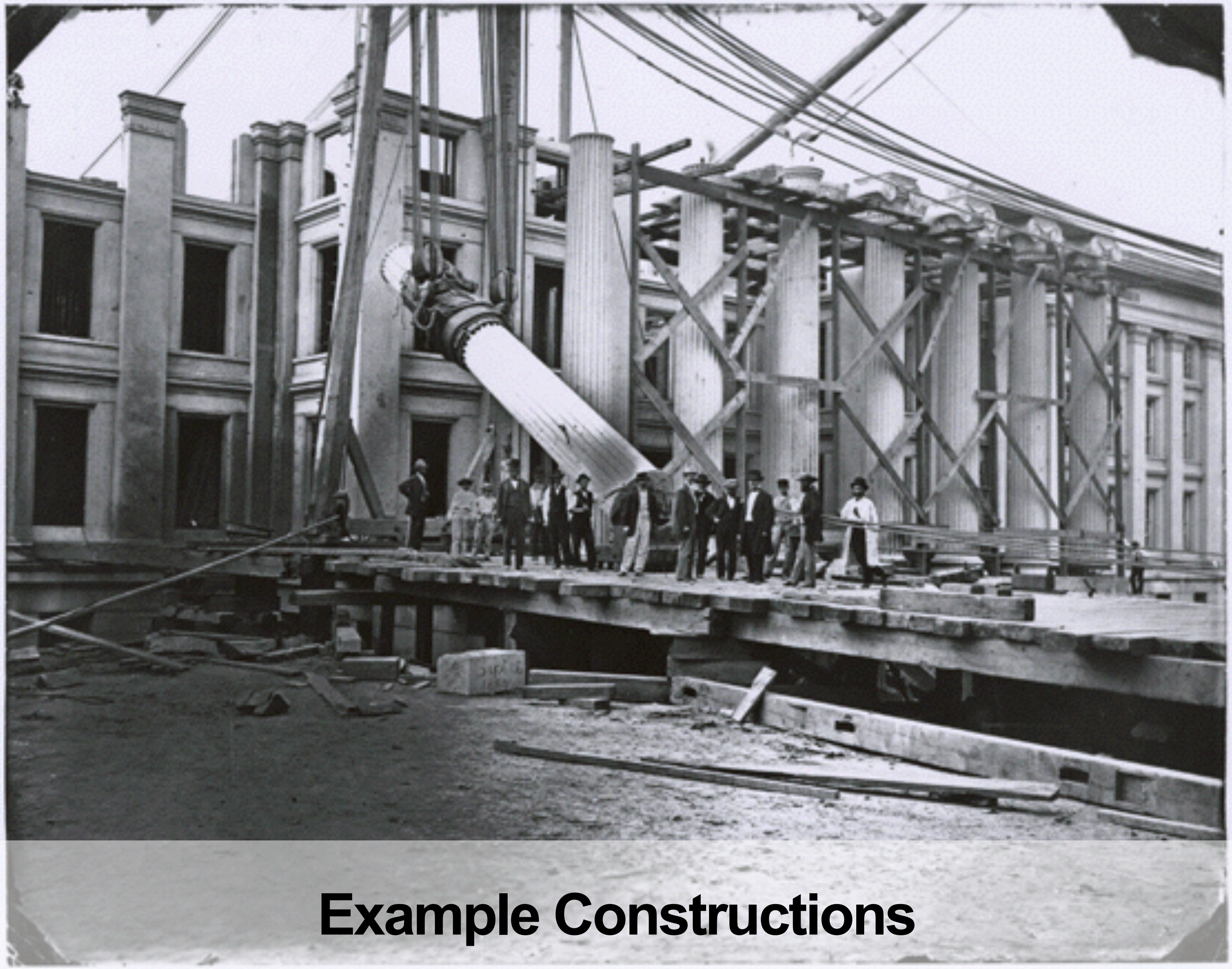
Prologue of the Code of Hammurabi, Louvre Museum
Photo by Marie-Lan Nguyen

Outline

- Example Constructions
- Learning from History
- The Model

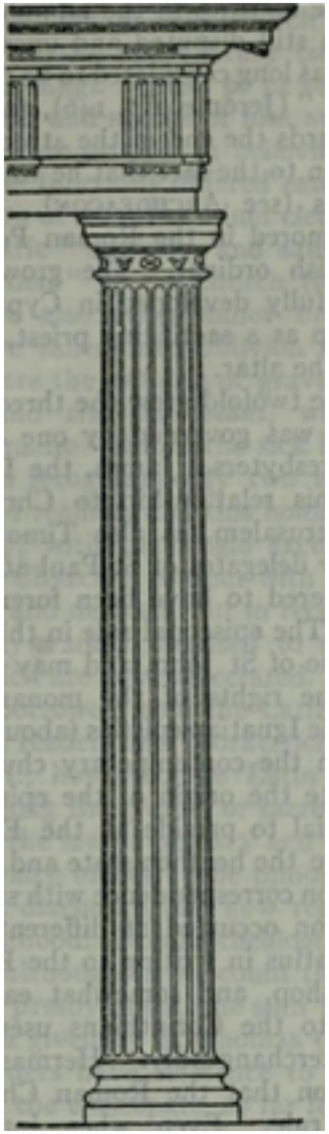
The Internet Archive: The Historian's History of the World

Example Constructions



US National Archives. Treasury Building construction

Design Goal 1



Internet Archive / 1910 Encyclopedia Britannica "The Italian Orders"

Sequential programs work in *program order*:

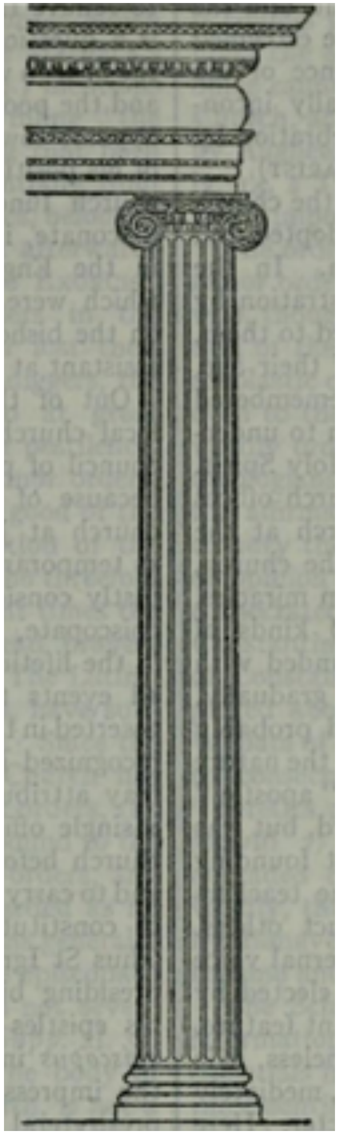
```
  var x:int;
  x = 1;
  x = 2;
  writeln(x);
```

should always output 2.

Note:

- CHARM++, OpenSHMEM don't follow this rule
- UPC, C, Java, Fortran do

Design Goal 2



Internet Archive / 1910 Encyclopedia Britannica "The Italian Orders"

Task constructs create additional dependencies:

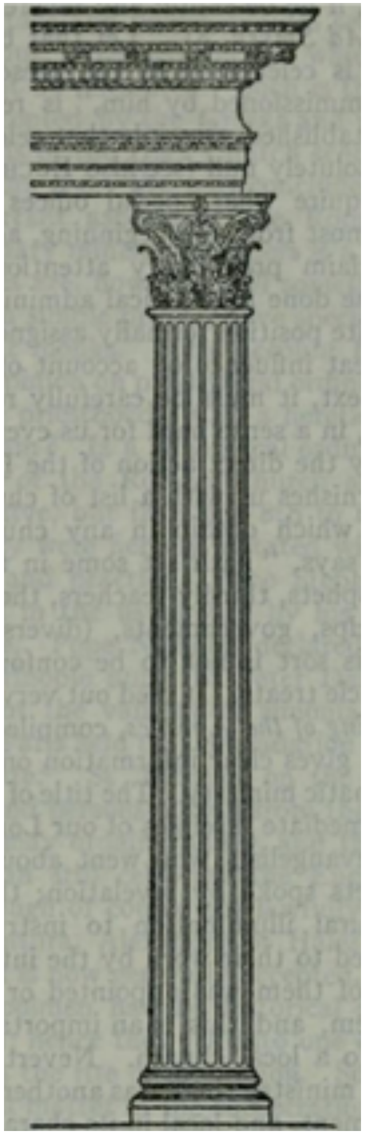
```
var x:int;
x = 50;
coforall i in 1..4 {
  writeln(x + i);
}
```

should always output a permutation of

51 52 53 54

in other words, x is always 50 in each task.

Design Goal 3



Internet Archive / 1910 Encyclopedia Britannica "The Italian Orders"

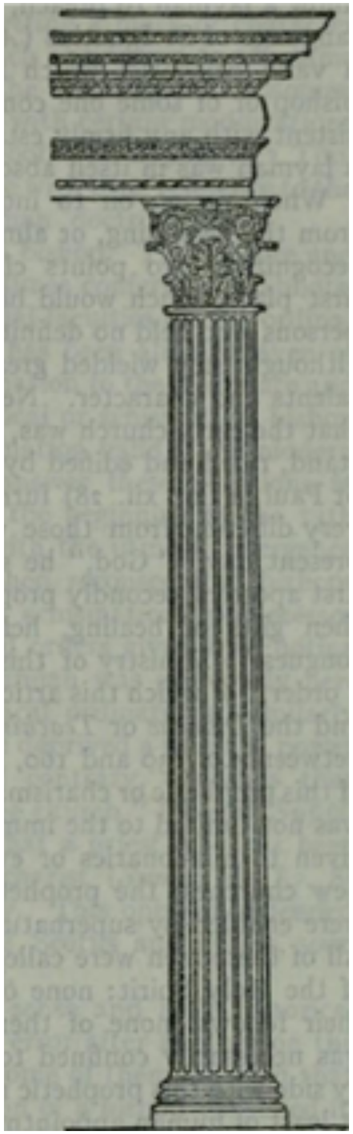
Remote memory has the same memory consistency rules as local memory:

```
var x: int;
on Locales[1] {
   x = 1;
   x = 2;
   writeln(x);
}
```

should always output 2.

Enables separation of algorithm from data layout.

Design Goal 4



Internet Archive / 1910 Encyclopedia Britannica "The Italian Orders"

The memory model should not inhibit common optimizations:

```
var x: int = 0;
cobegin ref(x) {
   { while x==0 {/*wait*/} }
   { x = 1; }
}
```

Has *undefined behavior* since there is a *data race* on variable x. Probably won't terminate.

In other words, the programmer must identify variables used to synchronize tasks. Need:

```
var x: atomic int;      or
var x: sync int;
```

Learning from History

Library of Congress

Learning from History

Library of Congress
Internet Archive - Natural History of the animal kingdom for the use of young people

Defining Racy Program Behavior

- Some specifications tried to define behavior for racy programs

→ inhibits optimization

→ usually wrong

- Java
  - circa 1996
  - fixed now
- UPC
  - attempted fix

Tasmanian Archives and Heritage Office
Leafy Sea Dragon sketch by William Buelow Gould

Impossible Implementation

- *shared strict* variables can synchronize processes

→ each *shared strict* variable must be atomic

- any type can be *shared strict*

→ a *shared strict* variable could be 64KB!

- but RDMA can't possible be atomic for a 64KB type!
- and *shared strict* casts to local ptr → no locks!

→ in practice, *shared strict* only works for small types

Tasmanian Archives and Heritage Office
Saw shark sketch by William Buelow Gould

SC for DRF: The Big House

C11, C++11, Java, UPC, Fortran 2008

Internet Archive
from Andrea Palladio, his life and works

Start with C++11 MCM

- At a high level: sequentially consistent behavior for data race free programs
- other things are possible with order= arguments for atomic operations
  - *relaxed*
  - *acquire*
  - *release ...*

Library of Congress

Enhance for Chapel

- local and remote data have same rules
- task constructs (e.g. cobegin) influence *program order*
- planned support for explicit *unordered* operations

flickr user Andy Hay, Temple of Vesta
CC by 2.0 https://creativecommons.org/licenses/by/2.0/

Questions?

```
var x:int;

x = 1;
x = 2;
writeln(x);
```

→ 2

```
var x:int;
x = 50;
coforall i in 1..4 {
  writeln(x + i);
}
```

→ permutation of 51 52 53 54

```
var x: int;
on Locales[1] {
  x = 1;
  x = 2;
  writeln(x); }
```

→ 2

```
var x: int = 0;
cobegin ref(x) {
  { while x==0 {} }
  { x = 1; }
}
```

→ undefined behavior

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA, and YARCDATA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

CRAY CHAPEL

CRAY
THE SUPERCOMPUTER COMPANY

http://chapel.cray.com chapel_info@cray.com https://github.com/chapel-lang/chapel/