The Exascale Programming Challenge and Chapel's Response

Brad Chamberlain, Chapel Team, Cray Inc.

SICM^2 Parallel Computing Workshop

March 29th, 2014

~~The Exascale Programming Challenge and Chapel's Response~~

Chapel, Life, the Universe*

Brad Chamberlain, Chapel Team, Cray Inc.

SICM^2 Parallel Computing Workshop

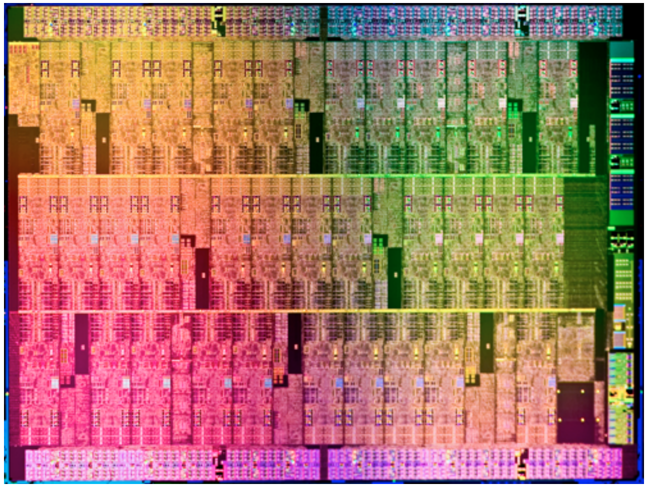
March 29th, 2014

* time permitting

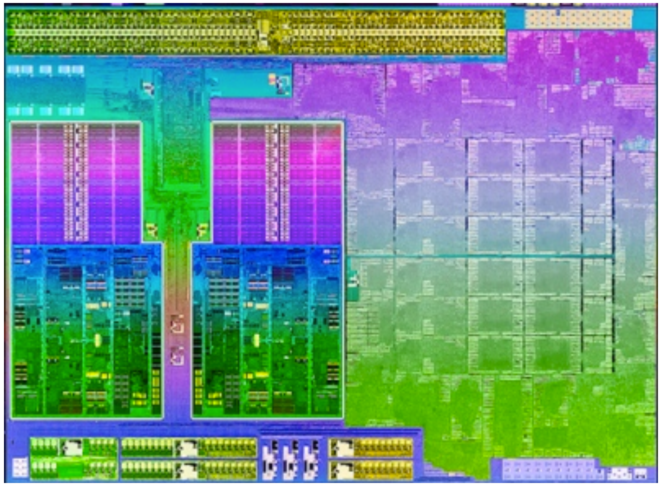
Safe Harbor Statement

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.

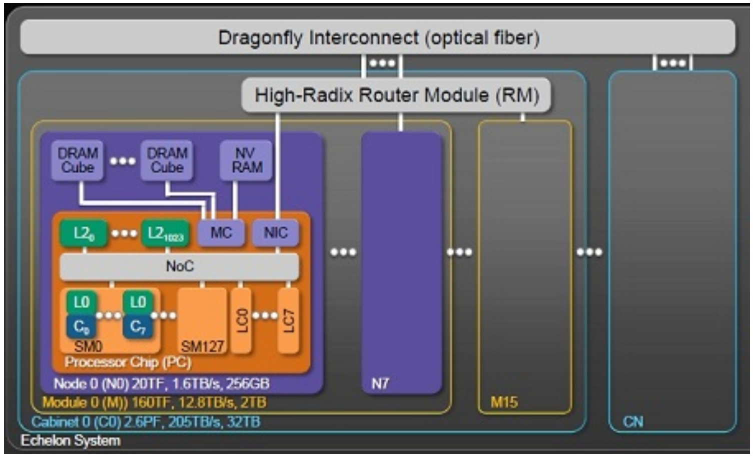
Prototypical Next-Gen Processor Technologies



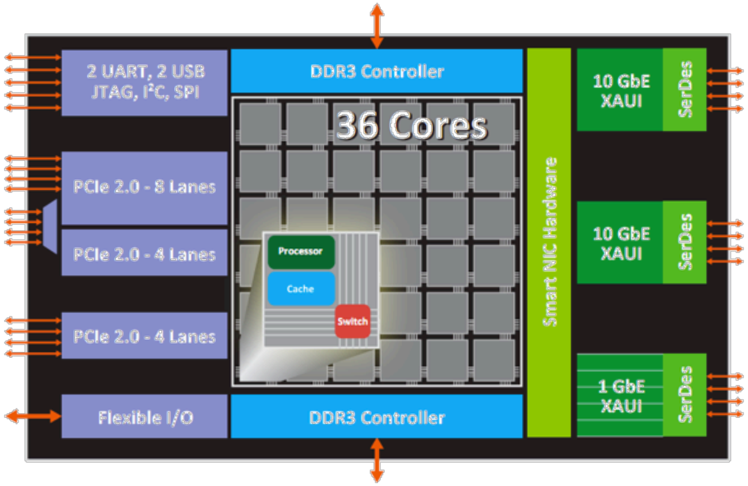
Intel MIC



AMD APU



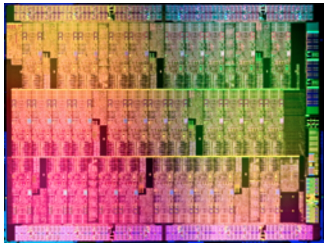
Nvidia Echelon

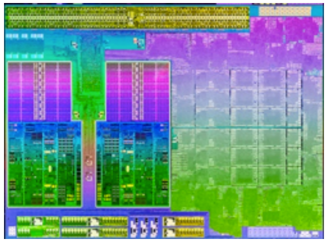


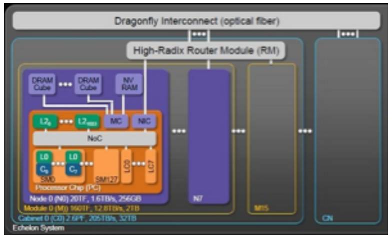
Tilera Tile-Gx

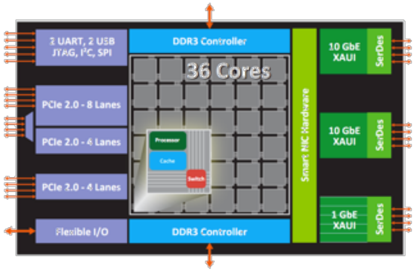
http://download.intel.com/pressroom/images/Aubrey_Isle_die.jpg http://www.zdnet.com/amds-trinity-processors-take-on-intels-ivy-bridge-3040155225/
http://insidehpc.com/2010/11/26/nvidia-reveals-details-of-echelon-gpu-designs-for-exascale/ http://tilera.com/sites/default/files/productbriefs/Tile-Gx%203036%20SB012-01.pdf

General Trends in These Architectures









- Increased hierarchy and/or sensitivity to locality
- Potentially heterogeneous processor/memory types

⇒ Next-gen programmers will have a lot more to think about at the node level than in the past

Why is there an exascale programming crisis?

Because HPC has adopted programming models that…

…have poor support for parallel work decomposition and scheduling

…have poor support for array layouts and distributed data structures

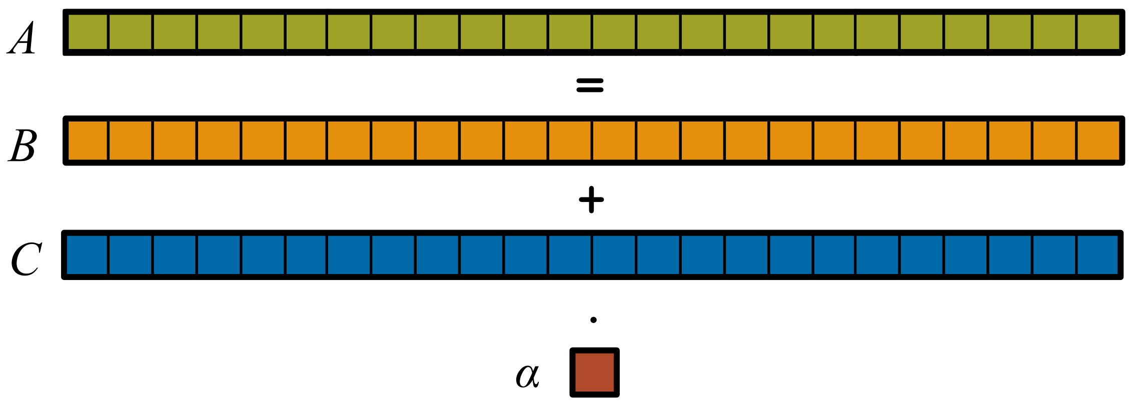
…tend to be closely tied to the architectural capabilities they target

STREAM Triad: a trivial parallel computation

Given: m-element vectors A, B, C

Compute: $\forall i \in 1..m,\ A_i = B_i + \alpha \cdot C_i$

In pictures:

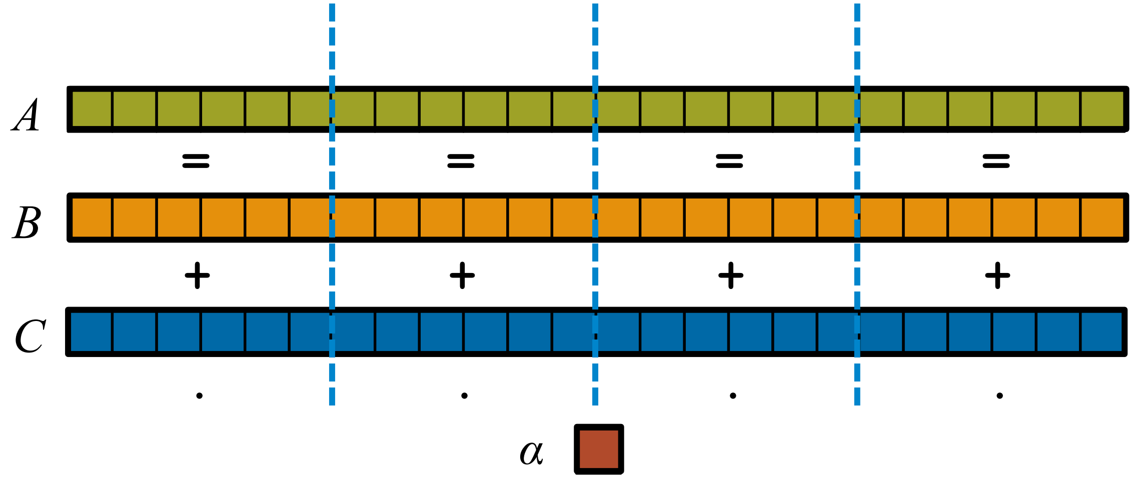


STREAM Triad: a trivial parallel computation

Given: m-element vectors A, B, C

Compute: $\forall i \in 1..m, A_i = B_i + \alpha \cdot C_i$

In pictures, in parallel:

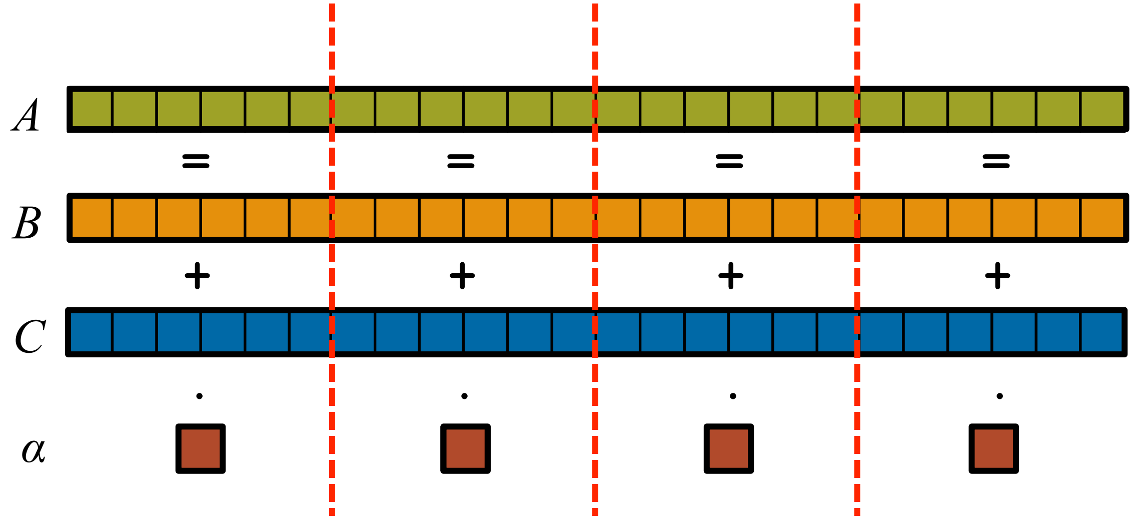


STREAM Triad: a trivial parallel computation

Given: m-element vectors A, B, C

Compute: $\forall i \in 1..m, A_i = B_i + \alpha{\cdot}C_i$

In pictures, in parallel (distributed memory):

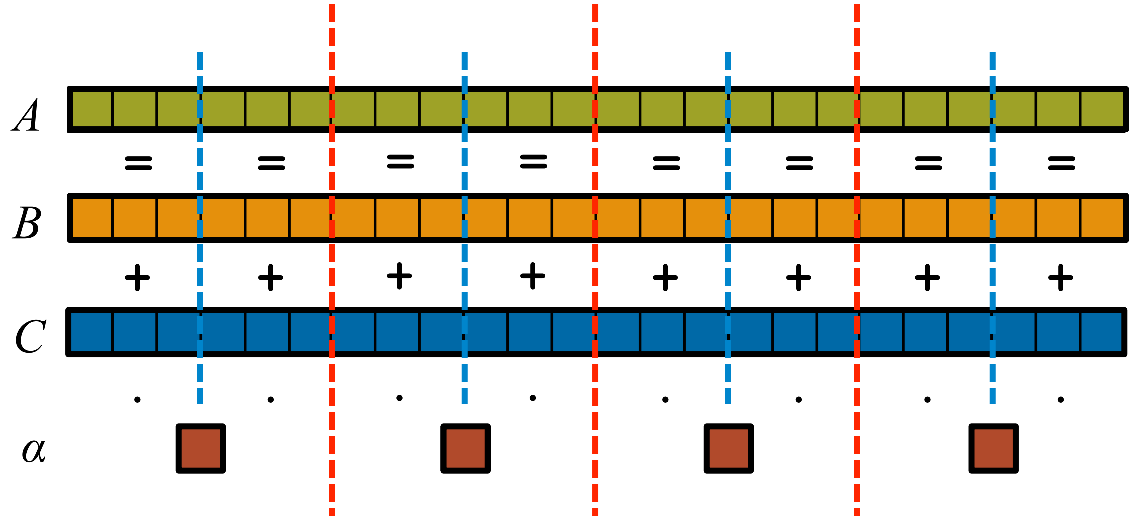


STREAM Triad: a trivial parallel computation

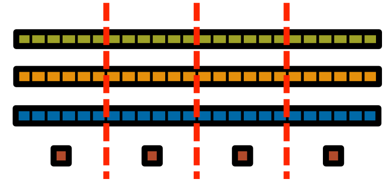
Given: m-element vectors A, B, C

Compute: $\forall i \in 1..m,\ A_i = B_i + \alpha \cdot C_i$

In pictures, in parallel (distributed memory multicore):



STREAM Triad: MPI

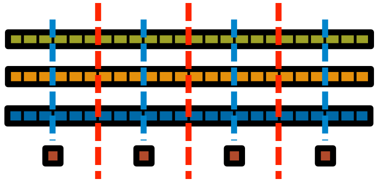


MPI

```
#include <hpcc.h>



static int VectorSize;
static double *a, *b, *c;

int HPCC_StarStream(HPCC_Params *params) {
  int myRank, commSize;
  int rv, errCount;
  MPI_Comm comm = MPI_COMM_WORLD;

  MPI_Comm_size( comm, &commSize );
  MPI_Comm_rank( comm, &myRank );

  rv = HPCC_Stream( params, 0 == myRank);
  MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM,
    0, comm );

  return errCount;
}

int HPCC_Stream(HPCC_Params *params, int doIO) {
  register int j;
  double  scalar;

  VectorSize = HPCC_LocalVectorSize( params, 3,
    sizeof(double), 0 );

  a = HPCC_XMALLOC( double, VectorSize );
  b = HPCC_XMALLOC( double, VectorSize );
  c = HPCC_XMALLOC( double, VectorSize );
```

```
  if (!a || !b || !c) {
    if (c) HPCC_free(c);
    if (b) HPCC_free(b);
    if (a) HPCC_free(a);
    if (doIO) {
      fprintf( outFile, "Failed to allocate memory (%d).
  \n", VectorSize );
      fclose( outFile );
    }
    return 1;
  }



  for (j=0; j<VectorSize; j++) {
    b[j] = 2.0;
    c[j] = 0.0;
  }

  scalar = 3.0;



  for (j=0; j<VectorSize; j++)
    a[j] = b[j]+scalar*c[j];

  HPCC_free(c);
  HPCC_free(b);
  HPCC_free(a);
```

STREAM Triad: MPI+OpenMP



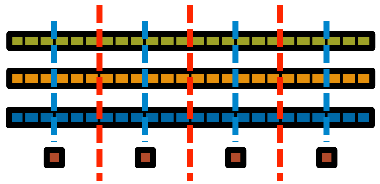
MPI + OpenMP

```
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

static int VectorSize;
static double *a, *b, *c;

int HPCC_StarStream(HPCC_Params *params) {
  int myRank, commSize;
  int rv, errCount;
  MPI_Comm comm = MPI_COMM_WORLD;

  MPI_Comm_size( comm, &commSize );
  MPI_Comm_rank( comm, &myRank );

  rv = HPCC_Stream( params, 0 == myRank);
  MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM,
    0, comm );

  return errCount;
}

int HPCC_Stream(HPCC_Params *params, int doIO) {
  register int j;
  double  scalar;

  VectorSize = HPCC_LocalVectorSize( params, 3,
    sizeof(double), 0 );

  a = HPCC_XMALLOC( double, VectorSize );
  b = HPCC_XMALLOC( double, VectorSize );
  c = HPCC_XMALLOC( double, VectorSize );
```

```
  if (!a || !b || !c) {
    if (c) HPCC_free(c);
    if (b) HPCC_free(b);
    if (a) HPCC_free(a);
    if (doIO) {
      fprintf( outFile, "Failed to allocate memory (%d).
     \n", VectorSize );
      fclose( outFile );
    }
    return 1;
  }

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++) {
    b[j] = 2.0;
    c[j] = 0.0;
  }

  scalar = 3.0;

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++)
    a[j] = b[j]+scalar*c[j];

  HPCC_free(c);
  HPCC_free(b);
  HPCC_free(a);
```

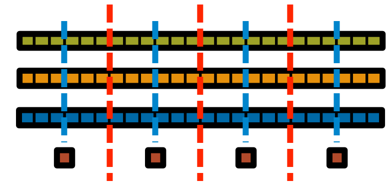
STREAM Triad: MPI+OpenMP vs. CUDA

MPI + OpenMP



```
#ifdef _OPENMP
#include <omp.h>
#endif

static int VectorSize;
static double *a, *b, *c;

int HPCC_StarStream(HPCC_Params *params) {
  int myRank, commSize;
  int rv, errCount;
  MPI_Comm comm = MPI_COMM_WORLD;

  MPI_Comm_size( comm, &commSize );
  MPI_Comm_rank( comm, &myRank );

  rv = HPCC_Stream( params, 0 == myRank);
  MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM, 0, comm );

  return errCount;
}

int HPCC_Stream(HPCC_Params *params, int doIO) {
  register int j;
  double  scalar;

  VectorSize = HPCC_LocalVectorSize( params, 3, sizeof(double), 0 );

  a = HPCC_XMALLOC( double, VectorSize );
  b = HPCC_XMALLOC( double, VectorSize );
  c = HPCC_XMALLOC( double, VectorSize );

  if (!a || !b || !c) {
    if (c) HPCC_free(c);
    if (b) HPCC_free(b);
    if (a) HPCC_free(a);
    if (doIO) {
      fprintf( outFile, "Failed to allocate memory (%d).\n", VectorSize );
      fclose( outFile );
    }
    return 1;
  }

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++) {
    b[j] = 2.0;
    c[j] = 0.0;
  }

  scalar = 3.0;

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++)
    a[j] = b[j]+scalar*c[j];

  HPCC_free(c);
  HPCC_free(b);
  HPCC_free(a);

  return 0;
}
```

CUDA



```
#define N        2000000

int main() {
  float *d_a, *d_b, *d_c;
  float scalar;

  cudaMalloc((void**)&d_a, sizeof(float)*N);
  cudaMalloc((void**)&d_b, sizeof(float)*N);
  cudaMalloc((void**)&d_c, sizeof(float)*N);

  dim3 dimBlock(128);
  if( N % dimBlock.x != 0 ) dimGrid

  set_array<<<dimGrid,dimBlock>>>(d_b, .5f, N);
  set_array<<<dimGrid,dimBlock>>>(d_c, .5f, N);

  scalar=3.0f;
  STREAM_Triad<<<dimGrid,dimBlock>>>(d_b, d_c, d_a, scalar,  N);
  cudaThreadSynchronize();

  cudaFree(d_a);
  cudaFree(d_b);
  cudaFree(d_c);
}

__global__ void set_array(float *a,  float value, int len) {
  int idx = threadIdx.x + blockIdx.x * blockDim.x;
  if (idx < len) a[idx] = value;
}

__global__ void STREAM_Triad( float *a, float *b, float *c,
                              float scalar, int len) {
  int idx = threadIdx.x + blockIdx.x * blockDim.x;
  if (idx < len) c[idx] = a[idx]+scalar*b[idx];
}
```

HPC suffers from too many distinct notations for expressing parallelism and locality

STREAM Triad: Chapel

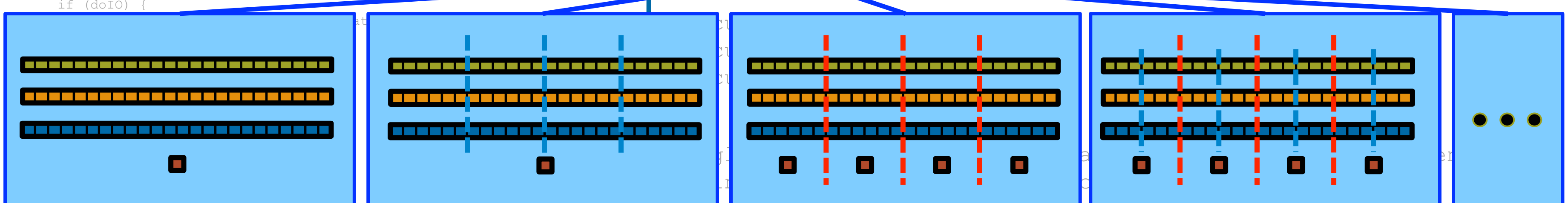
MPI + OpenMP

```
#ifdef _OPENMP
#include <omp.h>
#endif

static int VectorSize;
static double *a, *b, *c;

int HPCC_StarStream(HPCC_Params *par
  int myRank, commSize;
  int rv, errCount;
  MPI_Comm comm = MPI_COMM_WORLD;

  MPI_Comm_size( comm, &commSize );
  MPI_Comm_rank( comm, &myRank );

  rv = HPCC_Stream( params, 0 == myR
  MPI_Reduce( &rv, &errCount, 1, MPI

  return errCount;
}

int HPCC_Stream(HPCC_Params *params,
  register int j;
  double  scalar;

  VectorSize = HPCC_LocalVectorSize(

  a = HPCC_XMALLOC( double, VectorSi
  b = HPCC_XMALLOC( double, VectorSi
  c = HPCC_XMALLOC( double, VectorSi

  if (!a || !b || !c) {
    if (c) HPCC_free(c);
    if (b) HPCC_free(b);
    if (a) HPCC_free(a);
    if (doIO) {
```

Chapel

```
config const m = 1000,
             alpha = 3.0;

const ProblemSpace = {1..m} dmapped …;

var A, B, C: [ProblemSpace] real;

B = 2.0;
C = 3.0;

A = B + alpha * C;
```

the special sauce



Philosophy: Good language design can tease details of locality and parallelism away from an algorithm, permitting the compiler, runtime, applied scientist, and HPC expert to each focus on their strengths.

LULESH in Chapel

LULESH in Chapel

1288 lines of source code

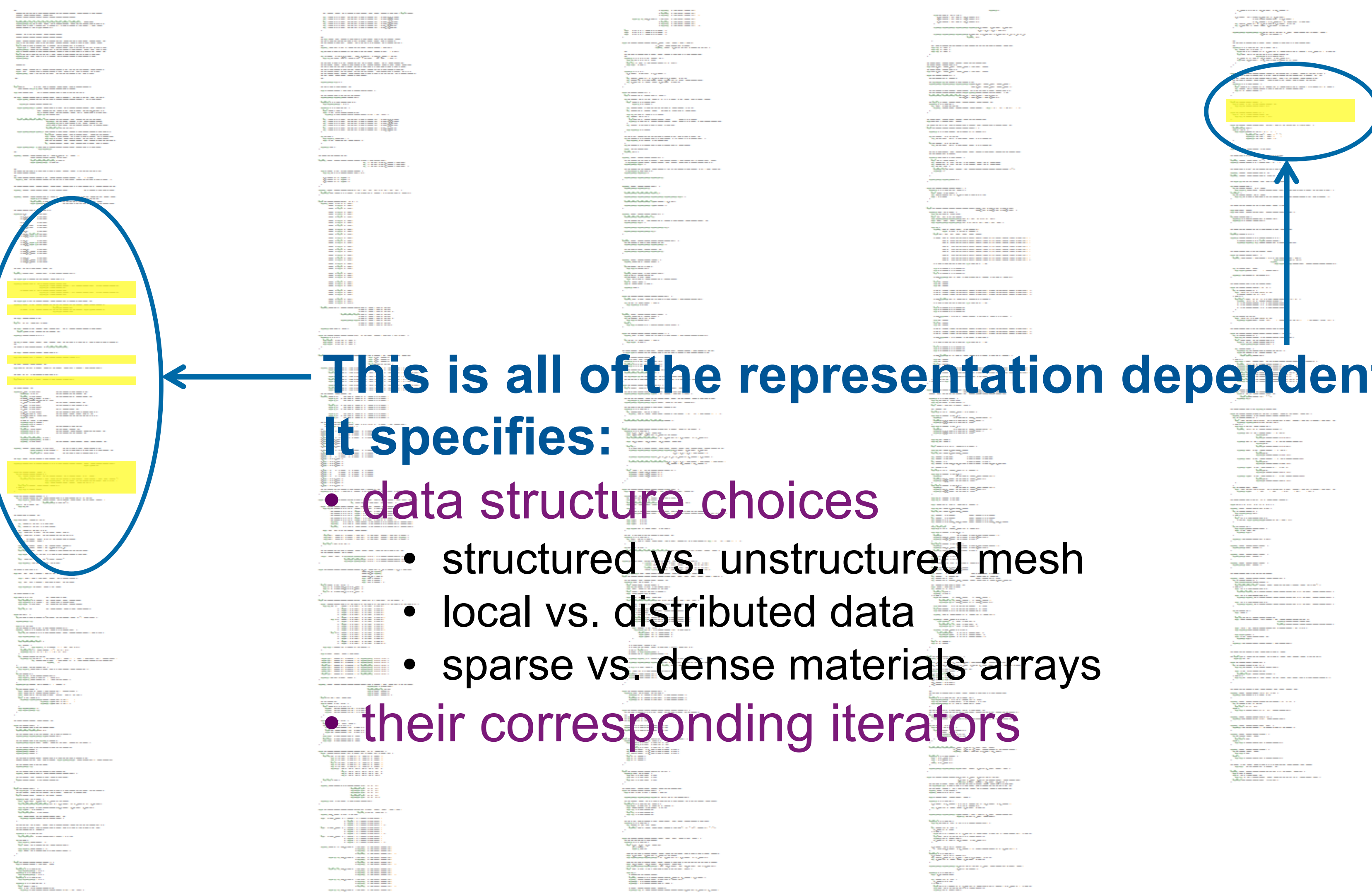
plus 266 lines of comments

487 blank lines

(the corresponding C+MPI+OpenMP version is nearly 4x bigger)

This is trunk/test/release/examples/benchmarks/lulesh/*.chpl in the SourceForge repository, as of r22745 (2/16/14).

LULESH in Chapel



This is all of the representation dependent code. It specifies:

- data structure choices
  - structured vs. unstructured mesh
  - local vs. distributed data
  - sparse vs. dense materials arrays
- their corresponding iterators

Why so many programming models?

HPC has traditionally given users…

…low-level, *control-centric* programming models
…ones that are closely tied to the underlying hardware
…ones that support only a single type of parallelism

Type of HW Parallelism	Programming Model	Unit of Parallelism
Inter-node	MPI	executable
Intra-node/multicore	OpenMP/pthreads	iteration/task
Instruction-level vectors/threads	pragmas	iteration
GPU/accelerator	CUDA/OpenCL/OpenACC	SIMD function/task

benefits: lots of control; decent generality; easy to implement
downsides: lots of user-managed detail; brittle to changes

What is Chapel?

- **An emerging parallel programming language**
  - Design and development led by Cray Inc.
    - in collaboration with academia, labs, industry
    - version 1.8 had19 contributors from 8 organizations and 5 countries
  - Initiated under the DARPA HPCS program

- **Being developed as open (BSD) software at SourceForge**

- **A work-in-progress**

Chapel's Targets

- **Target Architectures:**
  - multicore desktops and laptops
  - commodity clusters and the cloud
  - HPC systems from Cray and other vendors
  - *in-progress:* exascale-era architectures

- **Chapel's overall goal:** Improve programmer productivity

What does "Productivity" mean to you?

Recent Graduate:

"something similar to what I used in school: Python, Matlab, Java, …"

Seasoned HPC Programmer:

"that sugary stuff which I don't need because I ~~was born to suffer~~" [want full control/performance]

Computational Scientist:

"something that lets me focus on my parallel computational algorithms without having to wrestle with architecture-specific details"

Chapel Team:

"something that lets the computational scientist express what they want, without taking away the control an HPC programmer would want, implemented in a language as attractive as recent graduates want."

Three Chapel Successes

- **Effectively separating algorithms from system mappings**
  - user-defined array layouts and distributions
    - **alg:** "I'd like an array of this type over this index set"
    - **map:** "how should this array be distributed? stored locally?"
  - user-defined parallel iterators
    - **alg:** "forall …", whole-array operations, reductions, …
    - **map:** "how many tasks? how to divide the iterations?"
  - seamless integration of data and task parallelism
- **Distinct concepts for parallelism and locality***
  - "SPMD-only" and "shared memory-only" are restrictive to begin with
  - I believe they're non-starters in an exascale world
- **Withstanding the Naysayers**
  - we've generated cautious optimism in a community that's never had a productive language; and that has seen many, many failed attempts

(* I don't mean to suggest that Chapel was the first to do this—we weren't—simply that I believe it to be so crucial as to deserve silver)

Three Chapel Challenges

- **Performance**
  - the downside of permitting so much to be user-defined is that there's a bigger gap to close compared to the status quo

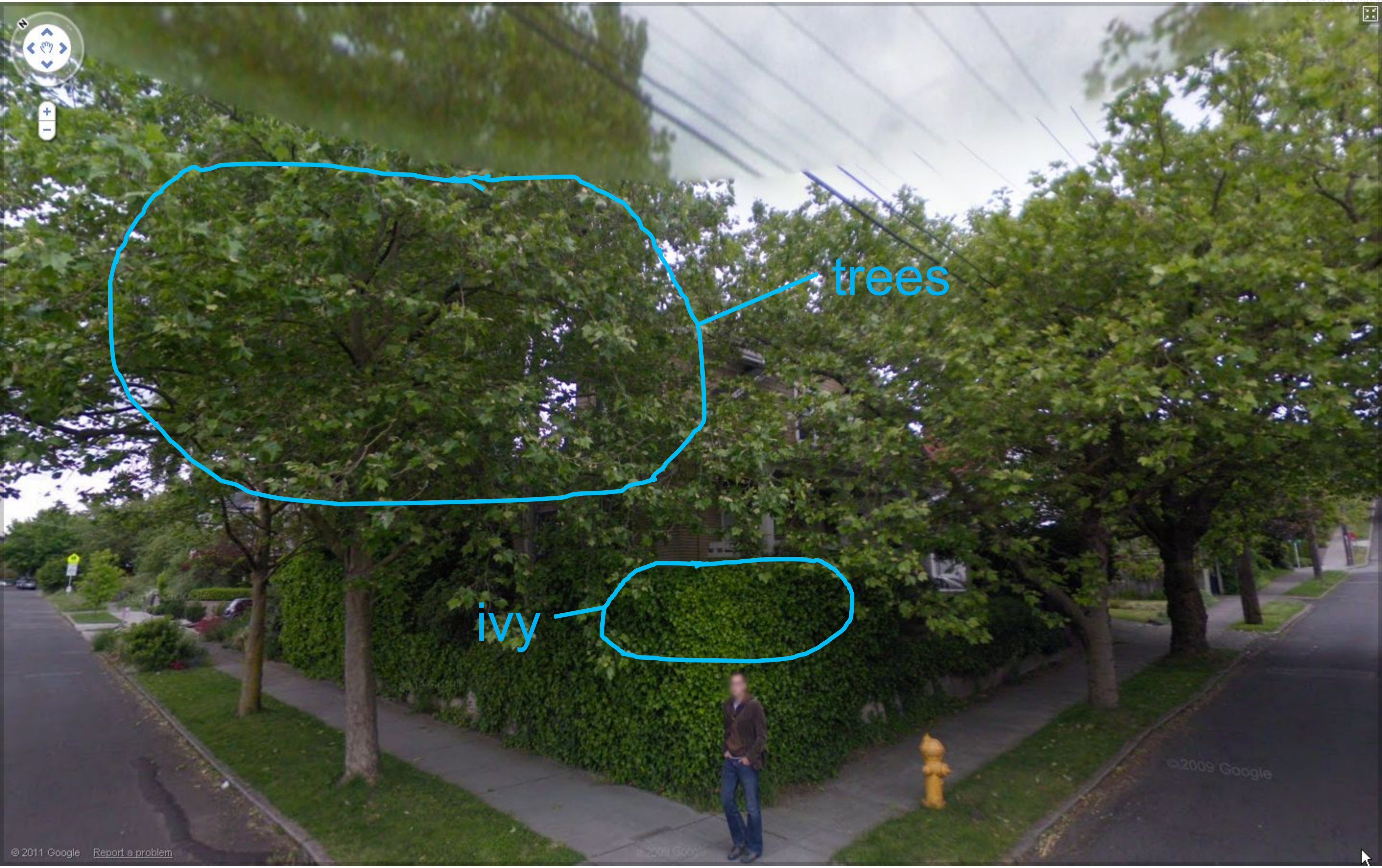
- **Reaching a Tipping Point in Acceptance/Utilization**
  - Chapel has lots of wallflower fans—how to get them invested?
  - and when?

- **Rapidly Responding to Emerging Architectures**
  - Chapel is designed to be forward portable, but effort is still required
  - ability to respond quickly would increase attractiveness

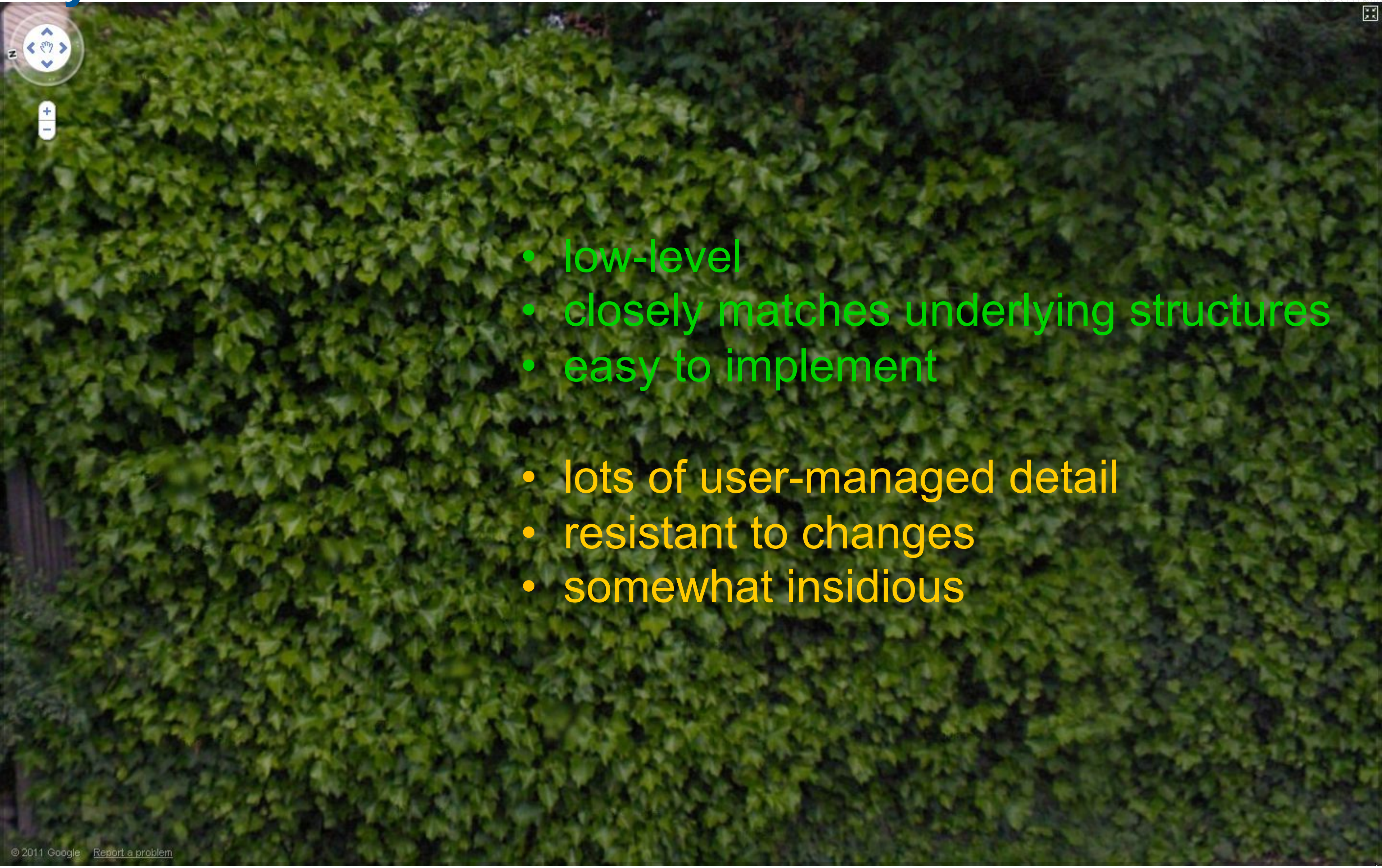
How Can Scientists Help?

- **Secure Time/Resources for Studying and Evaluating Promising Emerging Technologies**
  - no need to be more comprehensive than you desire
  - but if you don't like the status quo, invest some time in an alternative
- **Communicate your wishlists to new languages like Chapel**
  - "protect me from architectural changes" is a reasonable one
  - but surely you've got others?
- **Be patient**
  - no truly productive HPC-ready language is going to show up overnight without warning
- **Do something more constructive than stating the obvious**
  - yes, adoption of new languages is a difficult challenge
  - do you want to be part of the grumbling crowd, or part of the solution?

A Seattle Corner



Ivy



- low-level
- closely matches underlying structures
- easy to implement

- lots of user-managed detail
- resistant to changes
- somewhat insidious

Trees

- higher-level
- more elegant, structured

- requires a certain investment of time and force of will to establish

Landscaping Quotes from the HPC community

Early HPCS years:

"The HPC community tried to plant a tree once. It didn't survive. Nobody should ever bother planting one again."

"Why plant a tree if you can't be assured it will thrive?"

"Why would anyone ever want anything other than ivy?"

"We're in the business of building treehouses that last 40 years; we can't afford to build one in the branches of your sapling."

Landscaping Quotes from the HPC community

Early HPCS years (for the analogy-challenged):

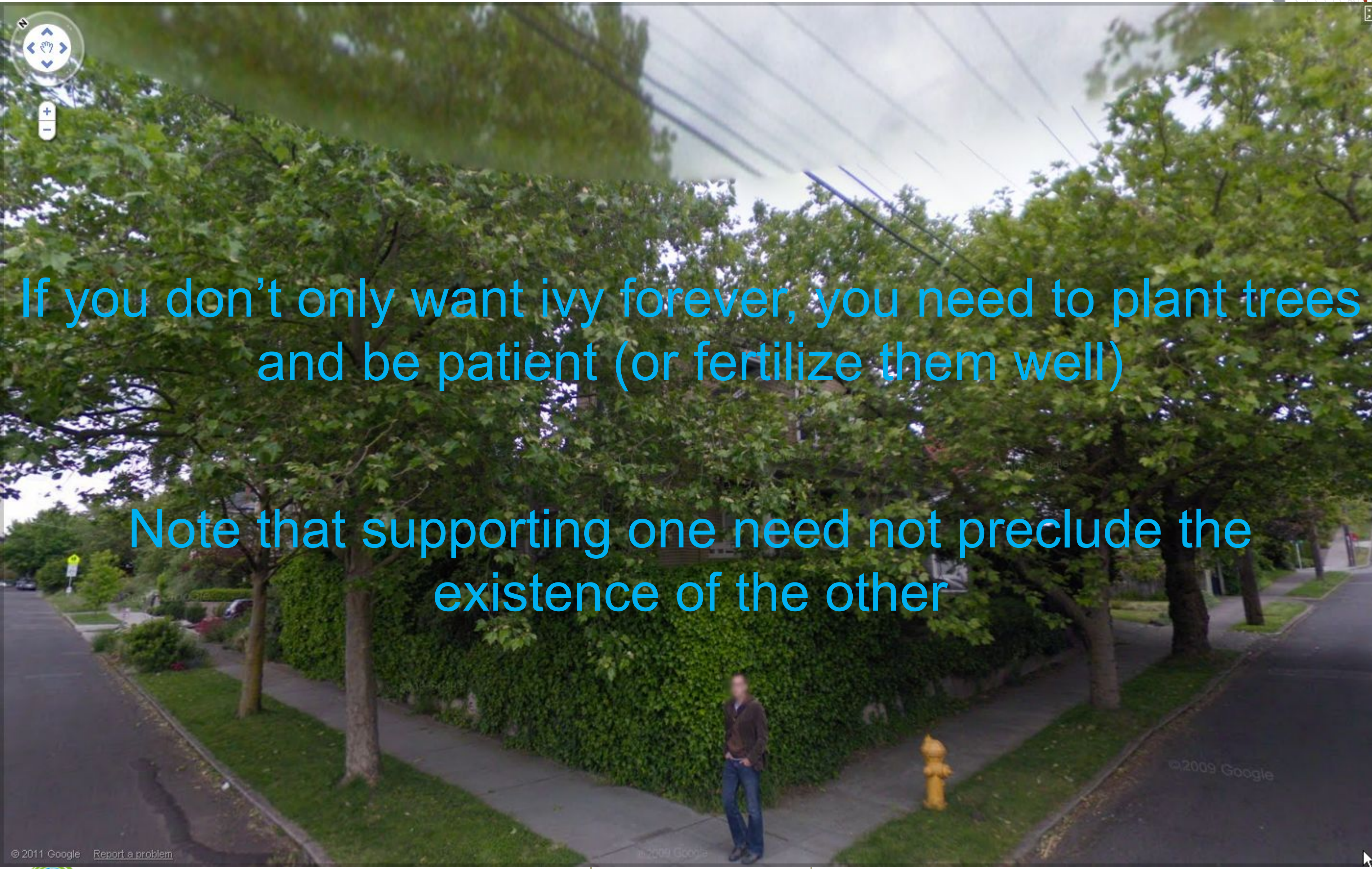
"The HPC community tried to develop a HLL once. It didn't survive. Nobody should ever bother developing one again."

"Why develop a language you can't be assured it will thrive?"

"Why would anyone ever want anything other than MPI+X?"

"We're in the business of writing applications that last 40 years; we can't afford to risk writing one in an emerging language."

A Corner in Seattle: Takeaways



If you don't only want ivy forever, you need to plant trees and be patient (or fertilize them well)

Note that supporting one need not preclude the existence of the other

Challenges for Computer Scientists

- **What are the abstractions that…**
  - give the application scientists the abstractions they want?
    - could be realized as DSLs, APIs, ADTs, …
  - support mapping down to multiple implementation choices
    - e.g., MPI+X as a safety net; Chapel as an investment in a better future
- **How do we collaborate effectively?**
  - there aren't many parallel language folks, and we each have our own
  - lone wolf researcher is seductive: independent, full control, full credit
    - but, we didn't reach the moon via dozens of partially-built rockets

A Note on Interoperability

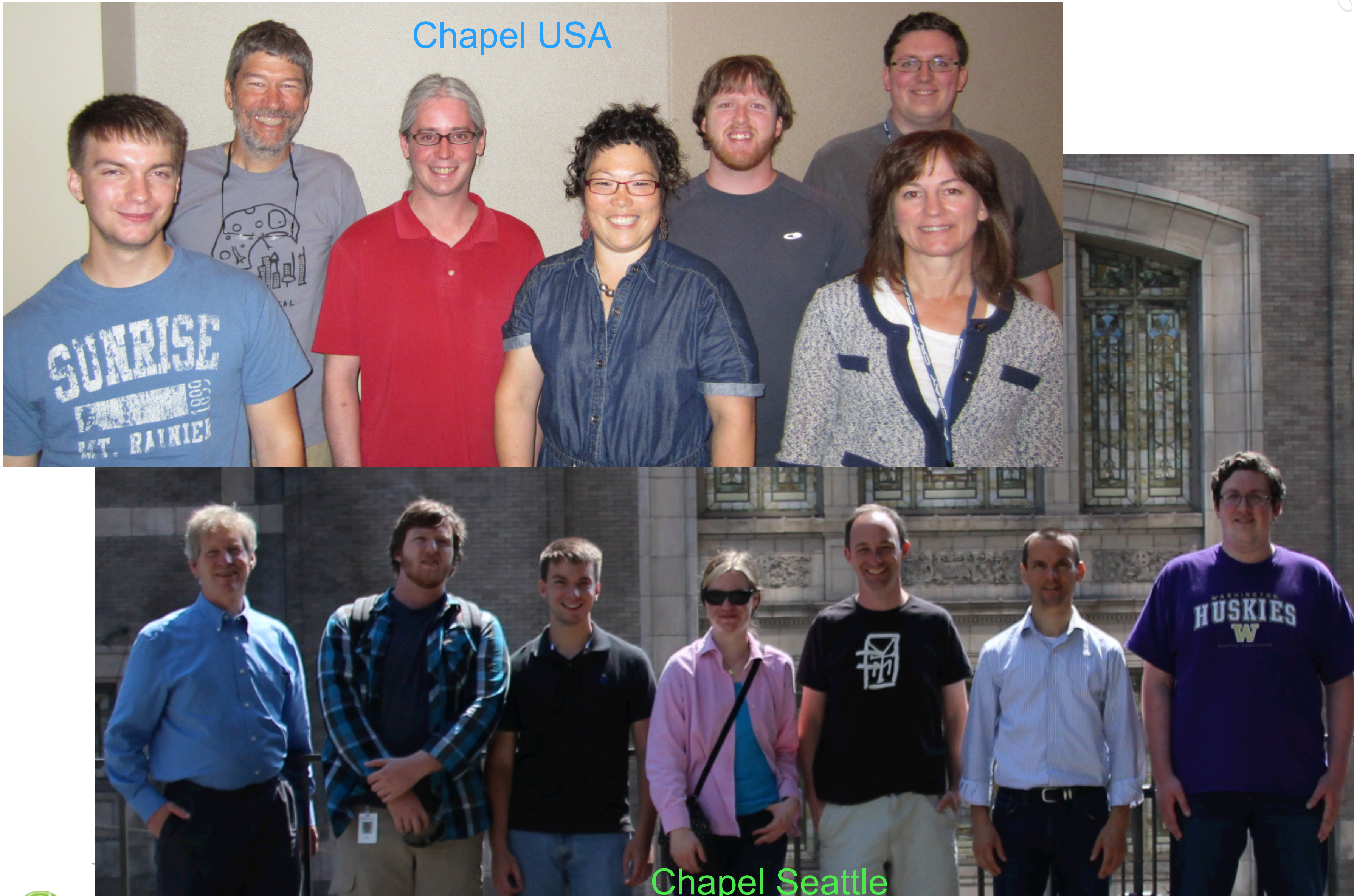
- If your language only supports one array format, it's only going to be efficiently interoperable with a small set of languages
- Via its user-defined array distributions and layouts, Chapel enables universal *in situ* interoperation

A Note on Parallel Education

- **When teaching parallel programming, I like to cover:**
  - data parallelism
  - task parallelism
  - concurrency
  - synchronization
  - locality/affinity
  - deadlock, livelock, and other pitfalls
  - performance tuning
  - …
- **I don't think there's been a good language out there…**
  - for teaching *all* of these things
  - for teaching some of these things well at all
  - ***until now:*** We believe Chapel can fill a crucial gap here

    (see http://chapel.cray.com/education.html for more information and http://cs.washington.edu/education/courses/csep524/13wi/ for my use of Chapel in class)

The Cray Chapel Team (Summer 2013)



Chapel is a collaborative effort… join us!

CRAY

LTS Laboratory for Telecommunications Sciences

RICE

Sandia National Laboratories

Lawrence Livermore National Laboratory

BERKELEY LAB
Lawrence Berkeley National Laboratory

Argonne NATIONAL LABORATORY

OAK RIDGE National Laboratory

Pacific Northwest NATIONAL LABORATORY
Proudly Operated by Battelle *Since 1965*

東京大学 THE UNIVERSITY OF TOKYO

ma UNIVERSIDAD DE MÁLAGA

UNIVERSITY OF MARYLAND

For More Information: Online Resources

Chapel project page: http://chapel.cray.com

- overview, papers, presentations, language spec, …

Chapel SourceForge page: https://sourceforge.net/projects/chapel/

- release downloads, public mailing lists, code repository, …

Mailing Aliases:

- chapel_info@cray.com: contact the team at Cray
- chapel-announce@lists.sourceforge.net: announcement list
- chapel-users@lists.sourceforge.net: user-oriented discussion list
- chapel-developers@lists.sourceforge.net: developer discussion
- chapel-education@lists.sourceforge.net: educator discussion
- chapel-bugs@lists.sourceforge.net: public bug forum

For More Information: Suggested Reading

Overview Papers:

- *The State of the Chapel Union* [slides], Chamberlain, Choi, Dumler, Hildebrandt, Iten, Litvinov, Titus. CUG 2013, May 2013.
  - *a high-level overview of the project summarizing the HPCS period*
- *A Brief Overview of Chapel*, Chamberlain (pre-print of a chapter for *A Brief Overview of Parallel Programming Models*, edited by Pavan Balaji, to be published by MIT Press in 2014).
  - *a more detailed overview of Chapel's history, motivating themes, features*

Blog Articles:

- *[Ten] Myths About Scalable Programming Languages*, Chamberlain. IEEE Technical Committee on Scalable Computing (TCSC) Blog, (https://www.ieeetcsc.org/activities/blog/), April-November 2012.
  - *a series of technical opinion pieces designed to rebut standard arguments against the development of high-level parallel languages*

Legal Disclaimer

CRAY CHAPEL

CRAY

THE SUPERCOMPUTER COMPANY

http://chapel.cray.com chapel_info@cray.com http://sourceforge.net/projects/chapel/