Exascale: An Opportunity to Atone for the Parallel Programming Models of the Past?

Brad Chamberlain, Cray Inc.

Punctuated Equilibrium at Exascale Panel/BoF

November 17th, 2011

v2

Disclaimer

This talk's contents should be considered my personal opinions (or at least one facet of them), not necessarily those of Cray Inc. nor my funding sources.

In many cases, when you hear "Chapel", feel free to mentally replace it with your favorite next-generation, non-evolutionary language.

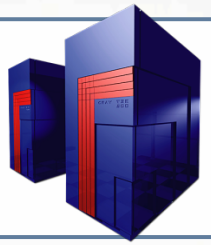
Sustained Performance Milestones

1 GF – 1988: Cray Y-MP; 8 Processors

- Static finite element analysis

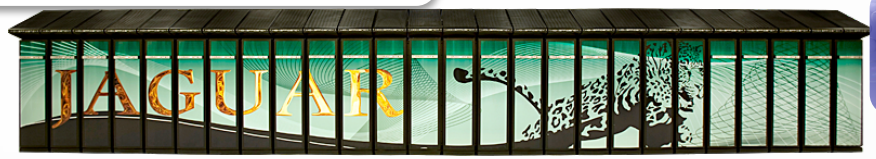
1 TF – 1998: Cray T3E; 1,024 Processors

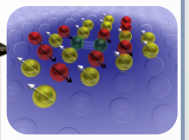
- Modeling of metallic magnet atoms



1 PF – 2008: Cray XT5; 150,000 Processors

- Superconductive materials





1 EF – ~2018: Cray ____; ~10,000,000 Processors

- TBD

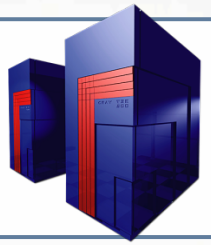
Sustained Performance Milestones

1 GF – 1988: Cray Y-MP; 8 Processors

- Static finite element analysis
- Fortran77 + Cray autotasking + vectorization

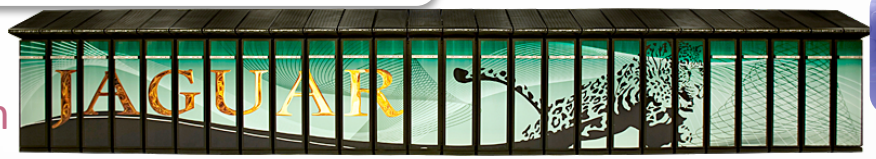
1 TF – 1998: Cray T3E; 1,024 Processors

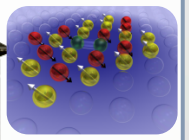
- Modeling of metallic magnet atoms
- Fortran + MPI (?)



1 PF – 2008: Cray XT5; 150,000 Processors

- Superconductive materials
- C++/Fortran + MPI + vectorization





1 EF – ~2018: Cray ____; ~10,000,000 Processors

- TBD
- TBD: C/C++/Fortran + MPI + CUDA/OpenCL/OpenMP/OpenACC

Or Perhaps Something Completely Different?

Q: Why Do HPC Programming Models Change?

HPC has traditionally given users…

…low-level, *control-centric* programming models

…ones that are closely tied to the underlying hardware

benefits: lots of control; decent generality; easy to implement

downsides: lots of user-managed detail; brittle to changes

A: Programming models have changed because our hardware architectures have changed and the software sits close enough to them that it too is forced to change

Change: It is Happening Again

- Exascale is expected to bring new changes/challenges:
  - increased hierarchy within the node architecture
    - *i.e.*, locality matters within a node, not just across nodes
  - increased heterogeneity as well
    - multiple processor types
    - multiple memory types
  - limited memory bandwidth, memory::FLOP ratio
  - resiliency concerns
  - power concerns

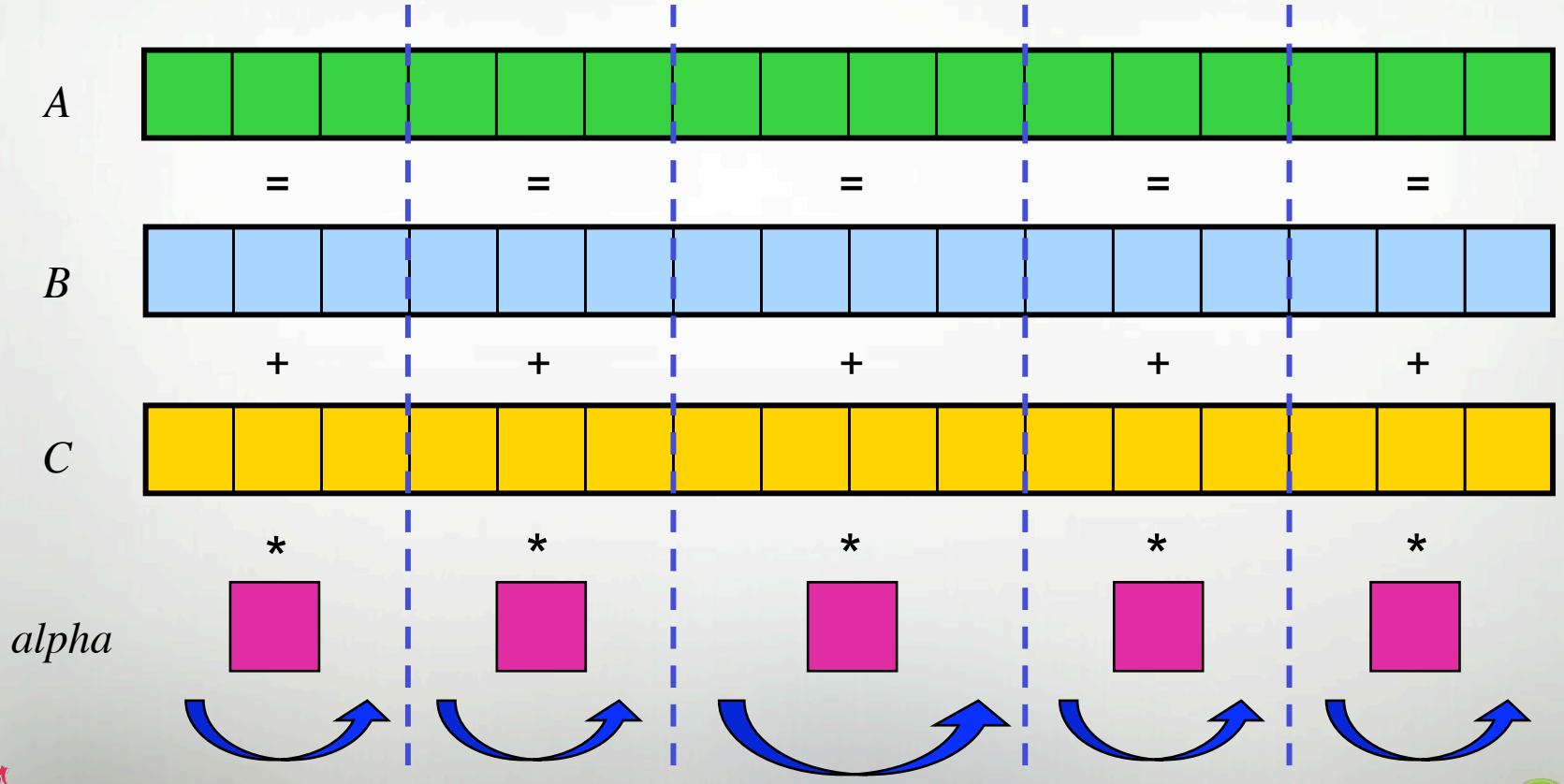
Exascale represents an opportunity to move to a programming model that is less tied to architecture than those of the past

Introduction to STREAM Triad

Given: m-element vectors A, B, C

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

Pictorially (in parallel):



A Few Versions of STREAM Triad

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);

  return 0;
}
```

A Few Versions of STREAM Triad

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);

  return 0;
}
```

A Few Versions of STREAM Triad

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);

  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);
  dim3 dimGrid(N/dimBlock.x );
  if( N % dimBlock.x != 0 ) dimGrid.x+=1;

  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

A Few Versions of STREAM Triad

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 == myRan
  MPI_Reduce( &rv, &errCount, 1, MPI_I

  return errCount;
}

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

  VectorSize = HPCC_LocalVectorSize( p

  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 all
      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;
```

```
;
;
;
```

```
N);
N);
```

```
d_c, d_a, scalar,  N);
```

```
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];
}
```

Chapel

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

const ProbSpace = [1..m] dmapped …;

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

B = …;
C = …;

A = B + alpha * C;
```

the special sauce

A Common Question

Q: Didn't we try this before with HPF?

Q': Orville, didn't Percy Pilcher die in *his* prototype powered aircraft?

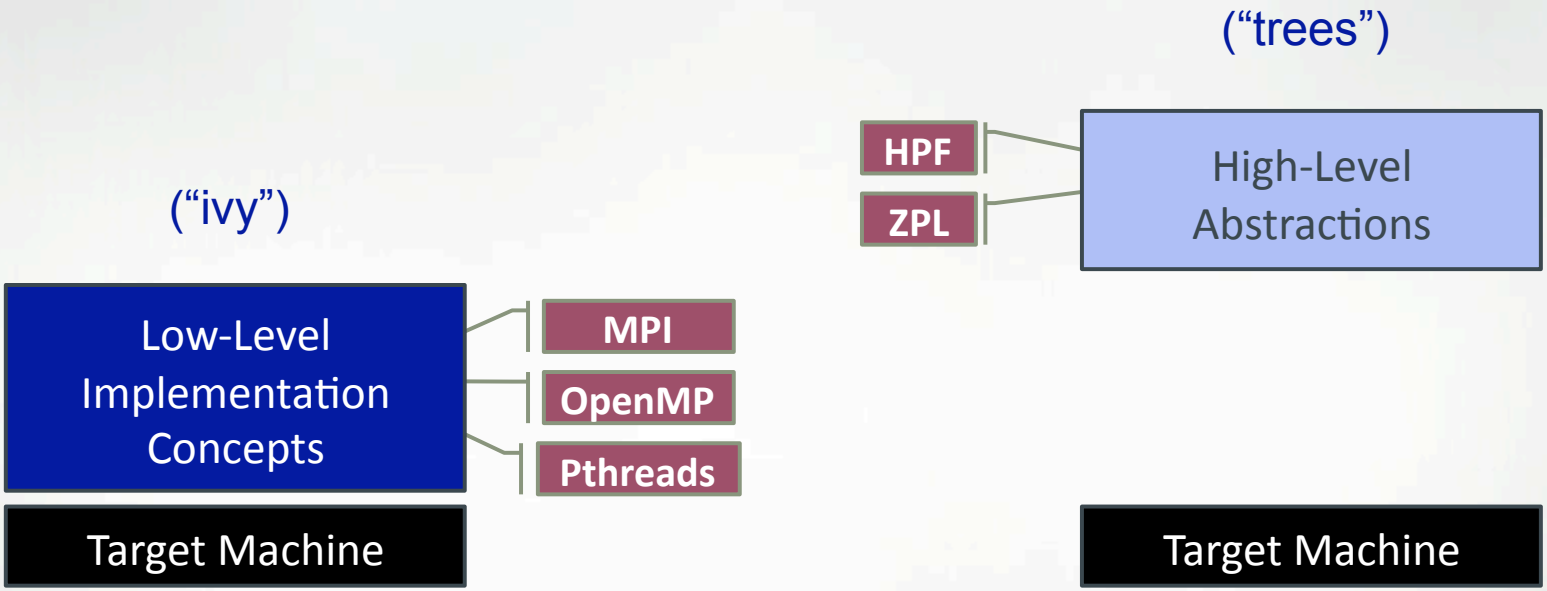
A': No Wilbur, he died in a glider; and even if it had been in his prototype, that doesn't mean we're doomed to fail.

Q: How Can Chapel Succeed When HPF Failed?

A: Chapel has had the chance to learn from HPF's mistakes (and other languages' successes and failures)

- Why did HPF fail?
  - lack of sufficient performance soon enough
  - vagueness in execution/implementation model
  - only supported a single level of data parallelism, no task/nested
  - inability to drop to lower levels of control
  - fixed set of limited distributions on dense arrays
  - lacked richer data parallel abstractions
  - lacked an open source implementation
  - too Fortran-based for modern programmers
  - …?
- The failure of one language---even a well-funded, US-backed one---does not dictate the failure of all future languages

Multiresolution Language Design: Motivation



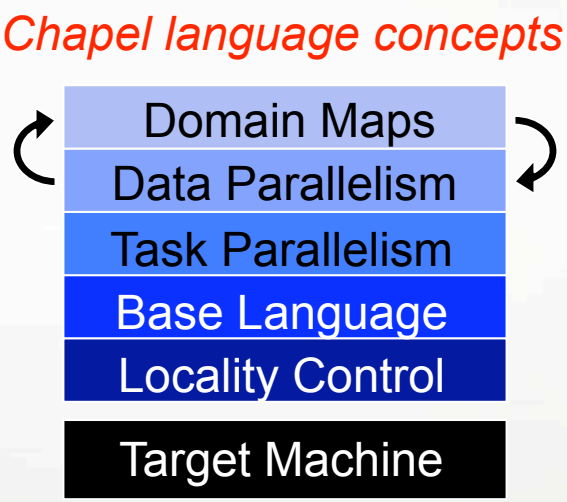
"Why is everything so tedious/difficult?"
"Why don't my programs port trivially?"

"Why don't I have more control?"

Chapel's Multiresolution Design

Multiresolution Design: Support multiple tiers of features

- higher levels for programmability, productivity
- lower levels for greater degrees of control



- build the higher-level concepts in terms of the lower
- permit the user to intermix layers arbitrarily

Global-View Need Not Preclude Control

A language can support both global- and local-view programming

```
proc main() {
  coforall loc in Locales do
    on loc do
      MySPMDProgram(loc.id, Locales.numElements);
}

proc MySPMDProgram(me, numCopies) {
  ...
}
```

Global-View Need Not Preclude Control

A language can support both global- and local-view programming (and even message passing)

```
proc main() {
  coforall loc in Locales do
    on loc do
      MySPMDProgram(loc.id, Locales.numElements);
}

proc MySPMDProgram(me, numCopies) {
  MPI_Reduce(mySumOfSquares, sumOfSquares,
             MPI_SUM, MPI_DOUBLE, 0,
             MPI_COMM_WORLD);
}
```

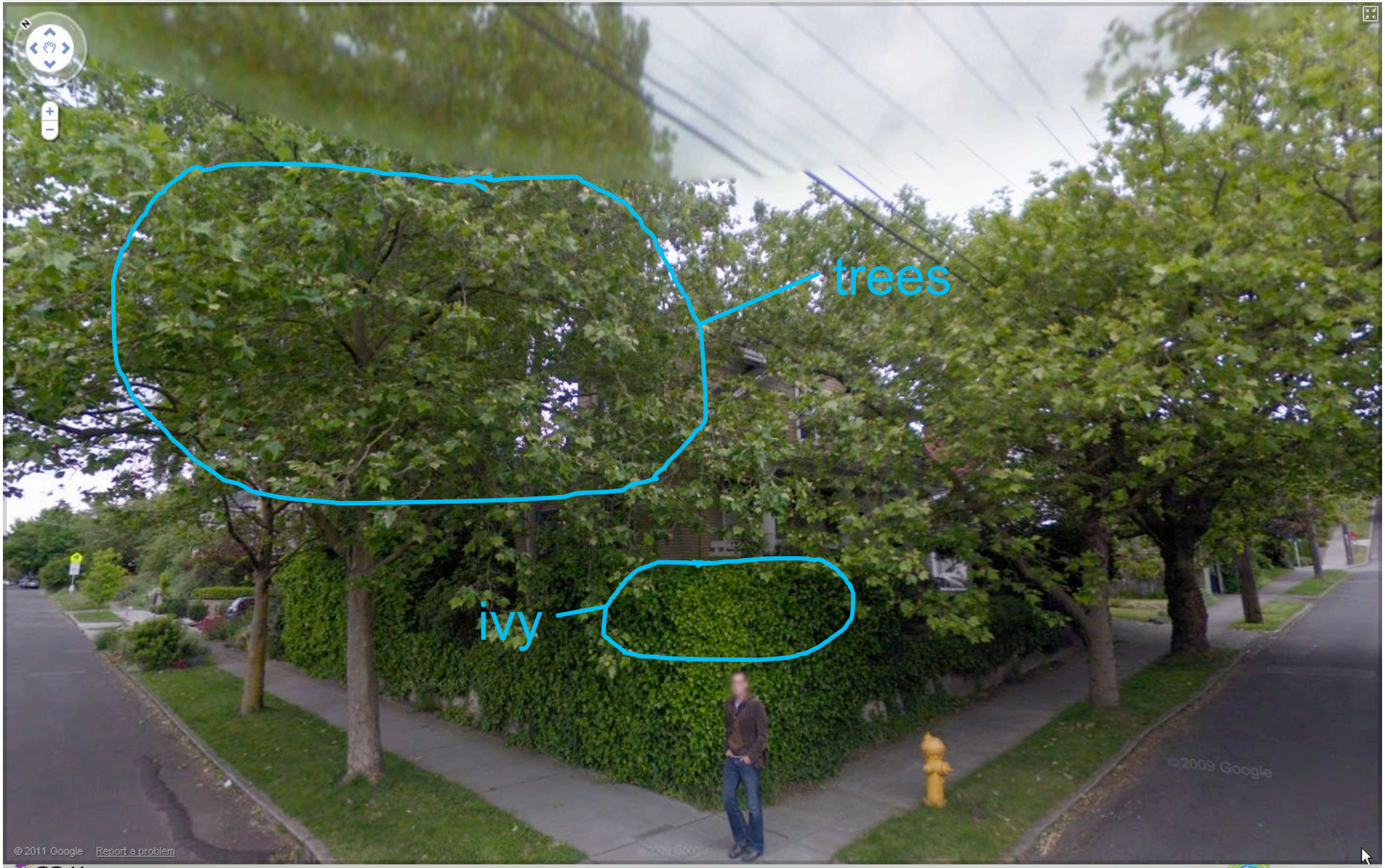
Chapel vs. HPF

Chapel ≠ HPF, due to its:

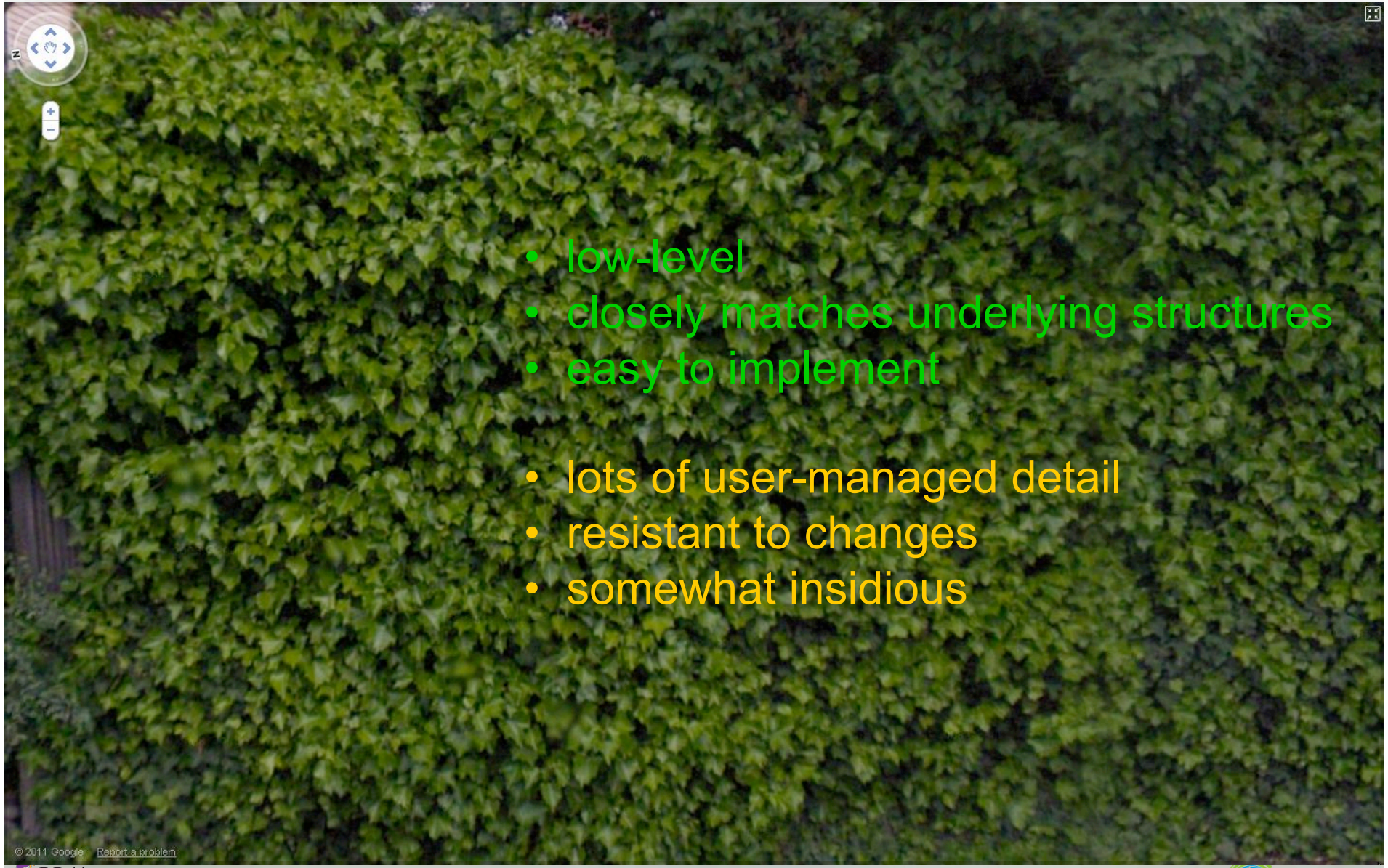
- ability to escape its global-view abstractions
- and other changes designed to address HPF's lacks:
  - well-defined execution model for data parallelism
  - support for task- and nested parallelism
  - user-defined distributions & layouts
  - open-source implementation
  - modern language design


CRAY
THE SUPERCOMPUTER COMPANY
SC11
Seattle, WA
Connecting communities through HPC
And now, a parable about landscaping…
CRAY
CHAPEL

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

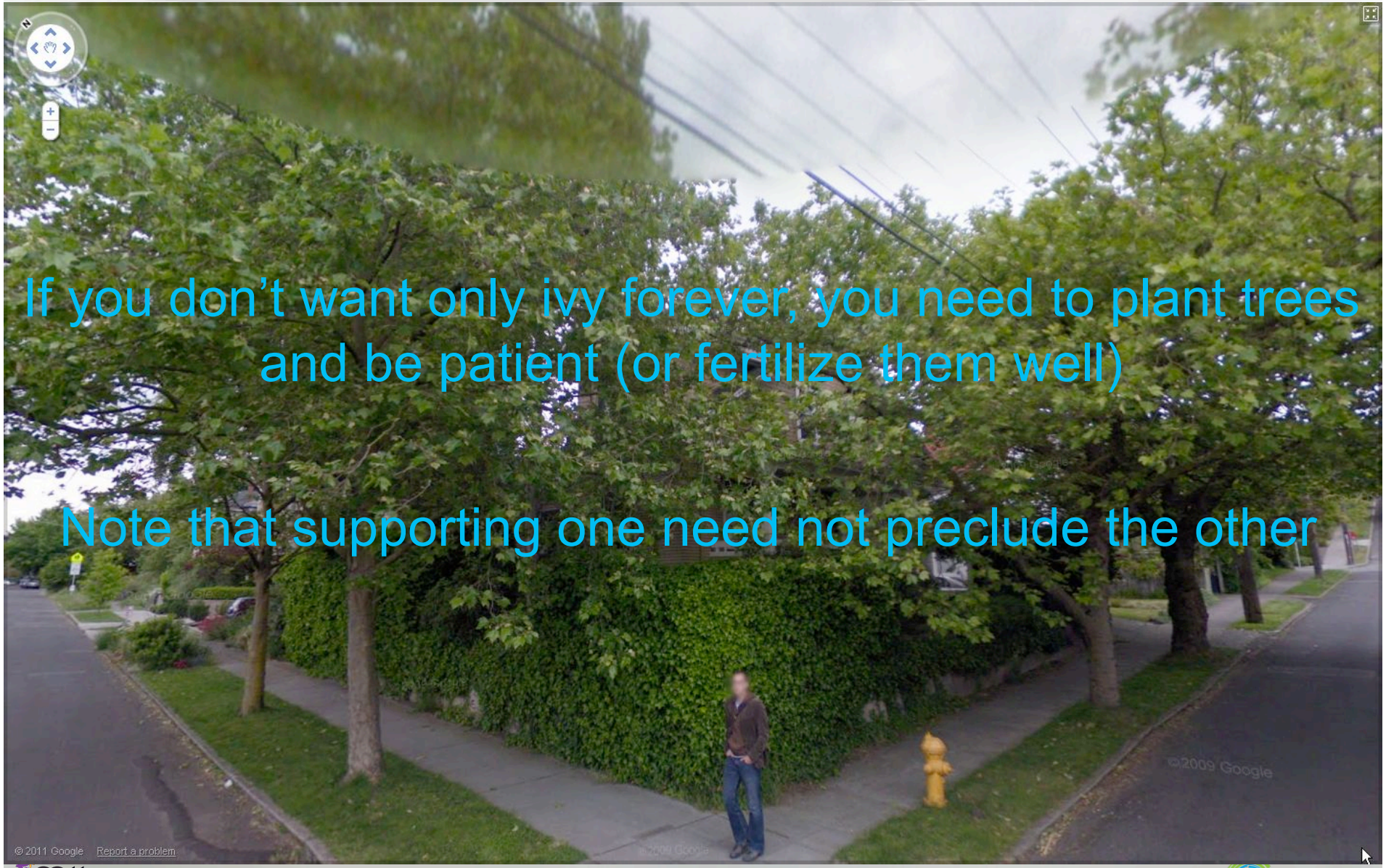
- requires a certain investment of time and force of will to establish

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 when you can't be assured it will grow?"
- "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."
- "This sapling looks promising. I'd like to climb it now!"

A Corner in Seattle: Takeaways



Pruning

- Planting many acorns can be wise
- Yet, not all acorns need to become trees
- Pruning is a healthy action as well
  - In particular, we should keep track of how new programming models might subsume old ones
- Mutations can be worse than planting a new sapling

So… do we rip out the ivy?

- Revolution rarely occurs in a vacuum

…nor overnight

- So, rather than asking questions like…
  - "Is the exascale programming model MPI+X?"
  - Or: "What is X?"

…we should instead be asking questions like "How could MPI best support an exascale revolution?"

  - Serving as middleware was, after all, part of its charter

My answer

- I think MPI still has an important role to play
  - to preserve legacy codes
  - to serve as a standard runtime/interoperability layer for revolutionary technologies
- But I also believe MPI needs to evolve:
  - Add excellent single-sided support
  - Support for active messages
  - To become more multithreading-oriented rather than process-centric

Summary

Higher-level programming models can help insulate science from implementation

- yet, without necessarily abandoning control
- Chapel does this via its multiresolution design

Exascale represents an opportunity to move to architecture-independent programming models

Past failures do not dictate future failures

Existing technologies will likely continue to play a role

- though perhaps not the one developers are accustomed to
- and not all existing technologies

CRAY

THE SUPERCOMPUTER COMPANY

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