High-Productivity Languages
for
Peta-Scale Computing

Hans P. Zima

Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA
and
University of Vienna, Austria
zima@jpl.nasa.gov

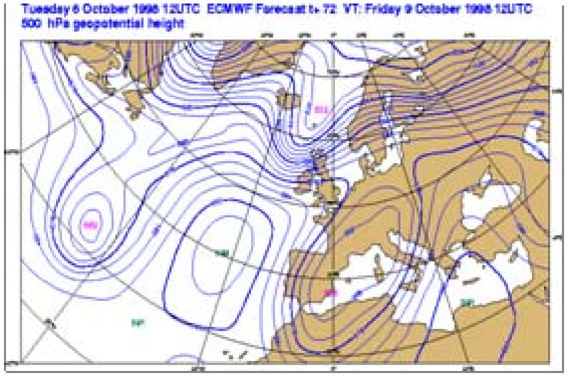
International Symposium on High Performance Distributed Computing (HPDC 2009)
Munich, Germany, June 13th, 2009

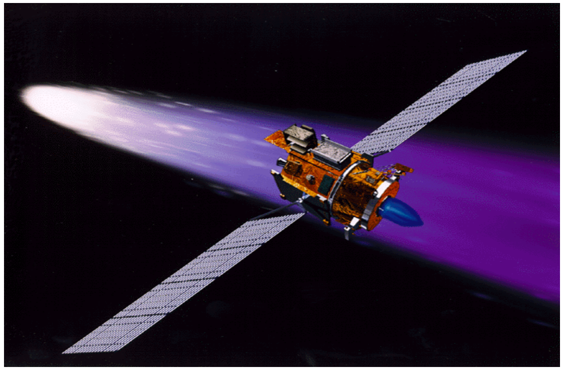
Contents

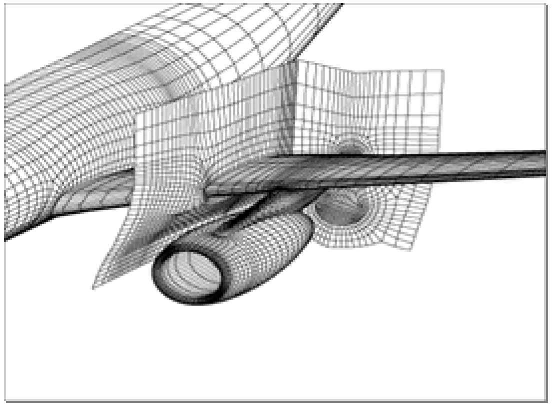
High Performance Computing has Become an Enabler of Progress in Science and Engineering

- **It constitutes the third pillar of science and engineering, in addition to *theory* and *experiment***
- **Traditional application areas include**
  - ***DNA Analysis***
  - ***Drug Design***
  - ***Medicine***
  - ***Aerospace***
  - ***Manufacturing***
  - ***Weather Forecasting and Climate Research***
- **New architectures provide new opportunities**
  - ***Graph Traversals***
  - ***Dynamic Programming***
  - ***Backtrack Branch & Bound***

UC Berkeley's "Dwarfs"







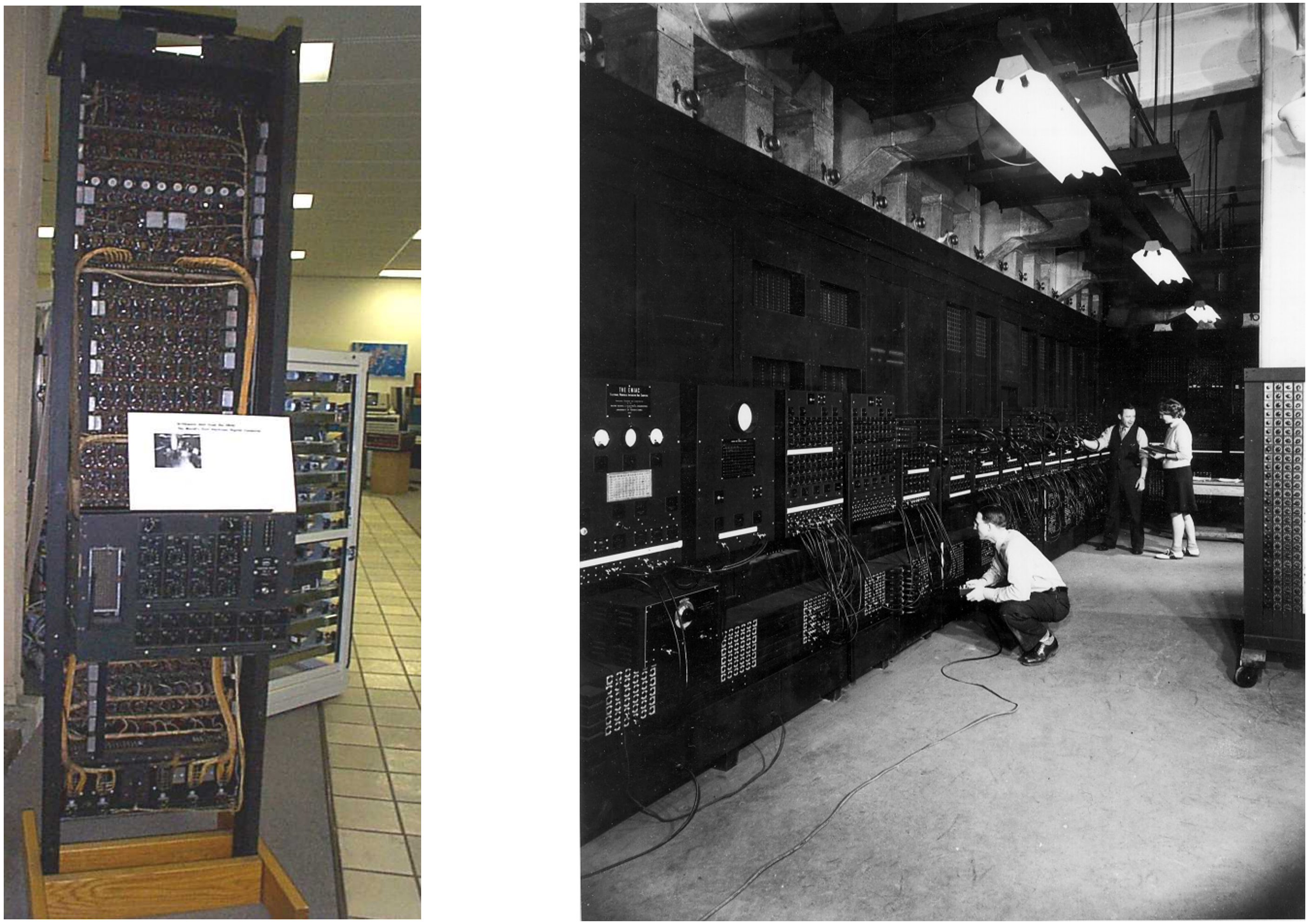
Hardware Development over 60 Years

This rise in the importance of HPC has happened in the context of a dramatic development of hardware technology over past decades:

- Performance growth:
  12 orders of magnitude

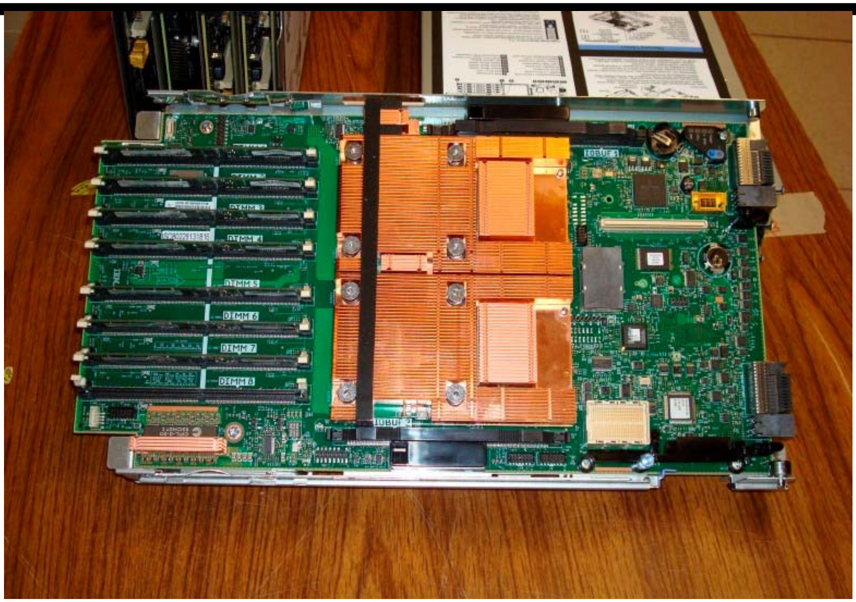
- Number of Processors:
  From 1 to more than 100,000

From Eniac (1946) …

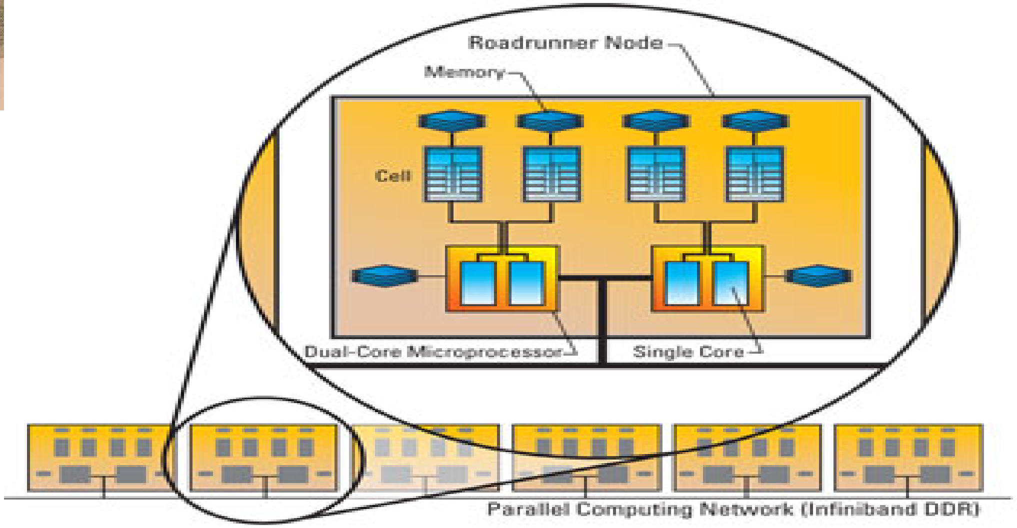


10^3 OPS

…to LANL Roadrunner: Top 500 #1



Cell Blade



1.105 PETAFLOPS

The first machine reaching Peta-scale performance

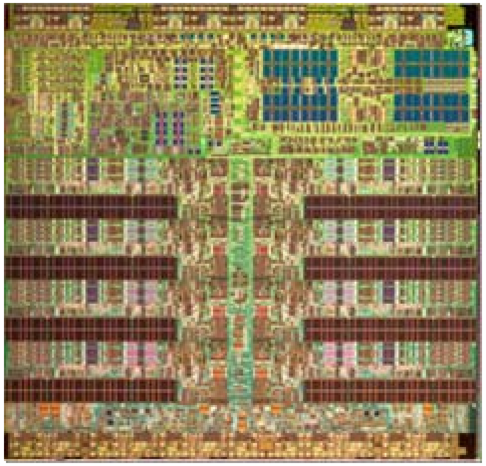
12,960 Cell chips (100 GF double precision)
Each Cell contains a PowerPC and 8 SPEs
6,480 dual-core Opterons
129,600 Cores
2,483 KW

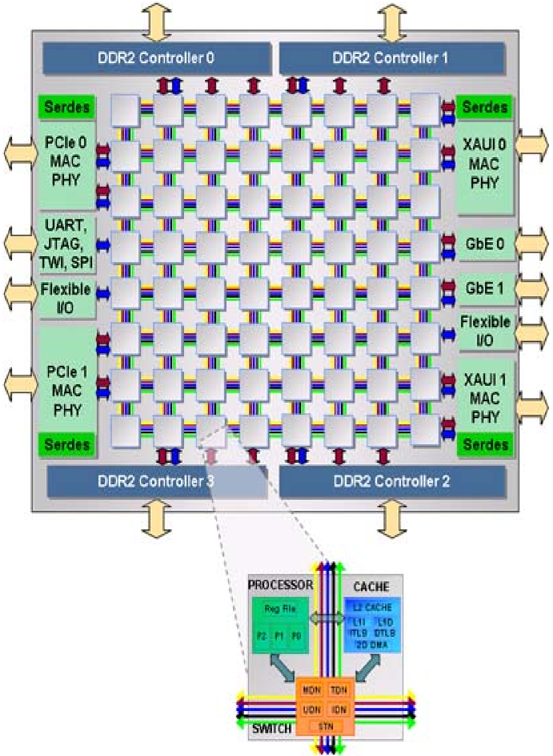
Reaching the Power Wall

- **1946-2004**
  - *general-purpose computing: sequential*
  - *clock frequency: 5 KHz → 4 GHz*
- **Since 2004**
  - *clock frequency growth is flat – as a result of power wall, instruction-level parallelism (ILP) wall*
  - *number of transistors per chip still grows exponentially*
  - *the only way to maintain exponential performance growth is parallelism*

Multi-Core Systems Dominating Computer Architectures

- **Cell Broadband Engine (IBM/Sony/Toshiba)**
  - *Power Processor (PPE) and 8 Synergistic PEs (SPEs)*
  - *peak 100 GF double precision (IBM Power XCEll 8i)*
- **Tile64 (Tilera Corporation, 2007)**
  - *64 identical cores, arranged in an 8X8 grid*
  - *iMesh on-chip network, 27 Tb/sec bandwidth*
  - *170-300mW per core; 600 MHz – 1 GHz*
  - *192 GOPS (32 bit)—about 10 GOPS/Watt*
- **Maestro: an RHBD version of Tile64 (2011)**
  - *49 cores, arranged in a 7X7 grid*
  - *70 GOPS at max power of 28W*
- **80-core research chip from Intel (2011)**
  - *2D on-chip mesh network for message passing*
  - *1.01 TF (3.16 GHz); 62W power—16 GOPS/Watt*
  - *Note: ASCI Red (1996): first machine to reach 1 TF*
    - *4,510 Intel Pentium Pro nodes (200 MHz)*
    - *500 KW for the machine + 500 KW for cooling of the room*





Contents

The Meaning of “High-Productivity”

◆ **“High productivity” implies three properties:**

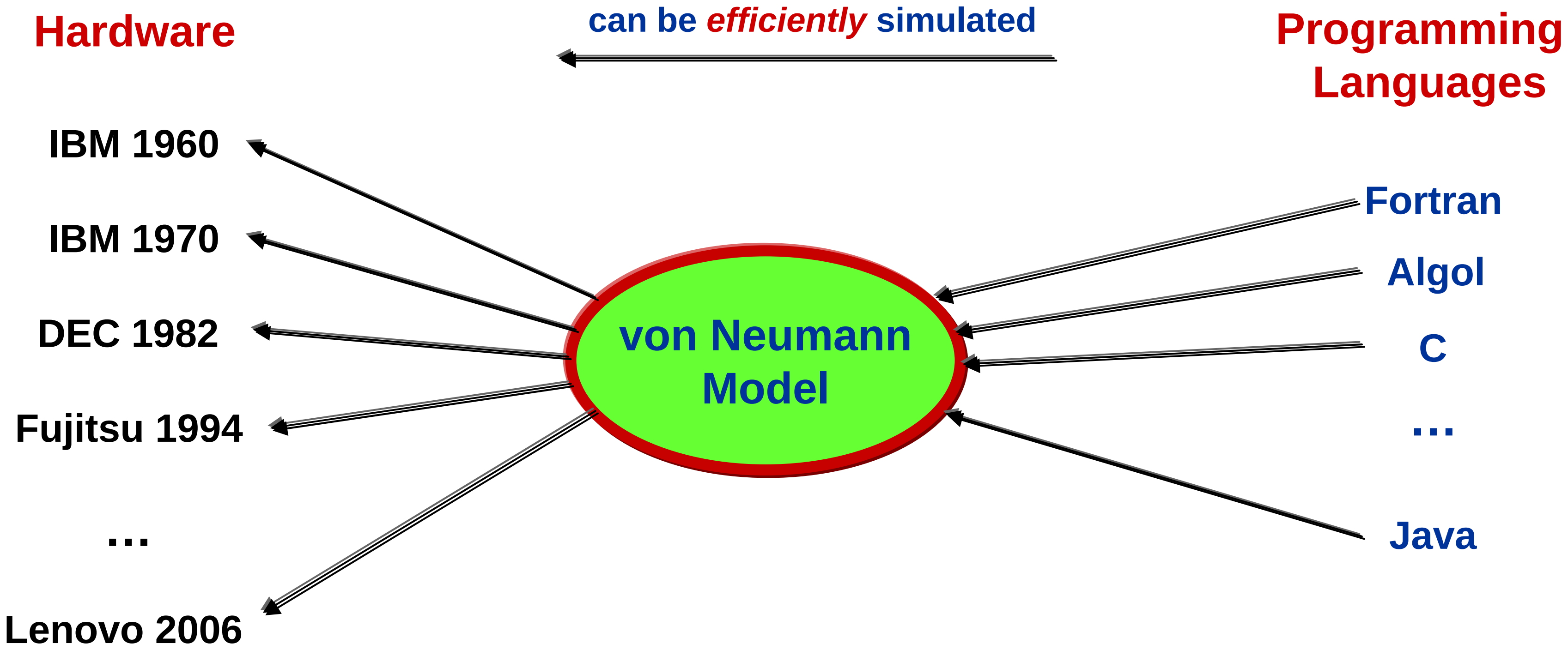
1. *human-centric: programming at a high level of abstraction*
2. *high-performance: providing “abstraction without guilt”*
3. *reliability*

◆ **Raising the level of abstraction is acceptable only if target code performance is not significantly reduced**

◆ **This relates to a broad range of topics:**

- *language design*
- *compiler technology*
- *operating and runtime systems*
- *library design and optimization*
- *intelligent tool development*
- *fault tolerance*

The Success of the von Neumann Model



The result of such a successful "bridging model" is performance portability: algorithms are written just once.

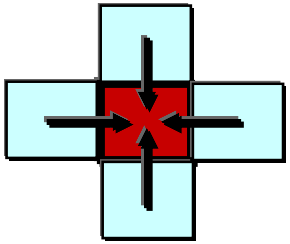
No comparable model has yet emerged for parallel programming. Efforts to find such a model began decades ago in the area of HPC…

MPI vs HPF:
An Example for Locality Management (Jacobi Relaxation)

```
real,  allocatable  A(:, : ), B(:, : )
      …
```

Sequential Code

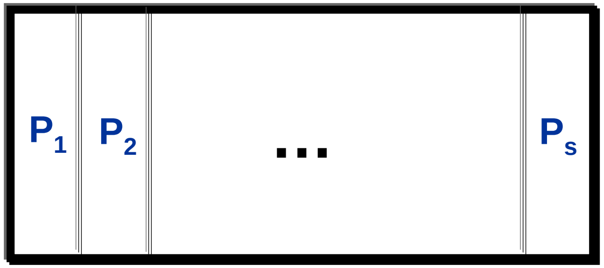
```
do while ( .not. converged )
   do J=1,N
        do I=1,N
             B(I,J)=0.25(A(I-1,J)+A(I+1,J)+A(I,J-1)+A(I,J+1))
         enddo
    enddo
A(1:N,1:N)=B
…
enddo
```



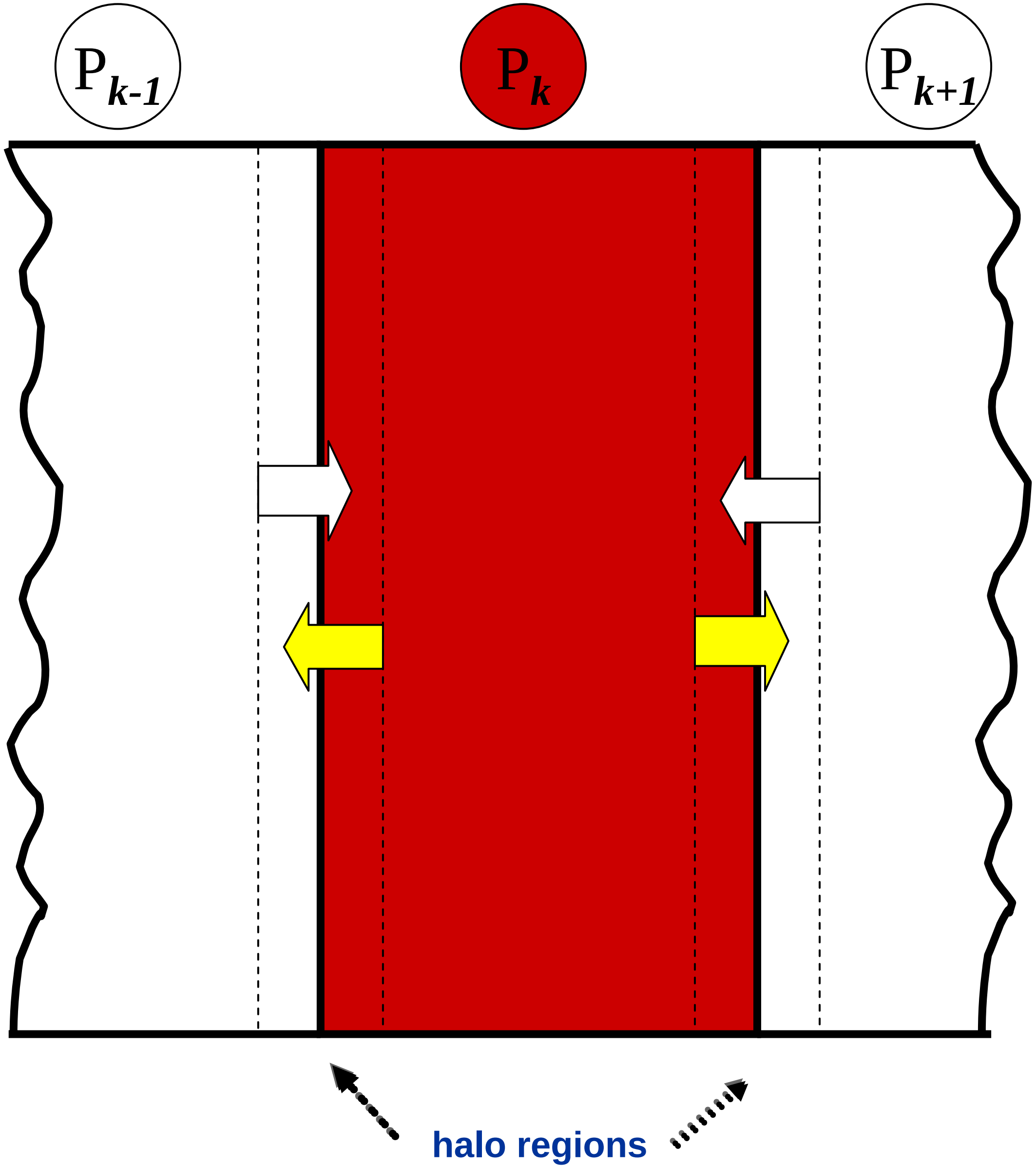
dependence pattern

Parallelization Based on Data Distribution

In a parallel code version, let A and B be partitioned into blocks of columns that are mapped to different processors. All these processors can work concurrently on their local data, but an exchange must take place after each iteration…



Boundary Exchange in Overlap Regions



```
! purely local operation in each iteration:
 do while ( .not. converged )
    do J=1,M   ! Number of local columns
       do I=1,N
          B(I,J)=0.25(A(I-1,J)+A(I+1,J)+
                      A(I,J-1)+A(I,J+1))
       enddo
    enddo
    …
```

After iteration:
Data Exchange

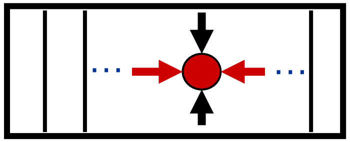
Processor P_k **reads:**

- *rightmost column* of P_{k-1}
- *leftmost column* of P_{k+1}.

Processor P_k **copies:**

- *its leftmost column* to P_{k-1}
- *its rightmost column* to P_{k+1}.

The Key Idea of High Performance Fortran (HPF)



Message Passing Approach

local view of data, **local** control,
explicit two-sided communication

initialize MPI

***local* computation**

```
do while (.not. converged)
    do  J=1,M
        do  I=1,N
             B(I,J) = 0.25 * (A(I-1,J)+A(I+1,J)+
                                          A(I,J-1)+A(I,J+1))
        end do
     end do
    A(1:N,1:N) = B(1:N,1:N)
```

```
if (MOD(myrank,2) .eq. 1) then
    call MPI_SEND(B(1,1),N,…,myrank-1,..)
    call MPI_RCV(A(1,0),N,…,myrank-1,..)
    if (myrank .lt. s-1) then
        call MPI_SEND(B(1,M),N,…,myrank+1,..)
        call MPI_RCV(A(1,M+1),N,…,myrank+1,..)
    endif
                                    else  …
                    …
```

HPF Approach

global view of data, **global** control,
compiler-generated communication

***global* computation**

```
do while (.not. converged)
    do  J=1,N
        do  I=1,N
             B(I,J) = 0.25 * (A(I-1,J)+A(I+1,J)+
                                          A(I,J-1)+A(I,J+1))
        end do
     end do
    A(1:N,1:N) = B(1:N,1:N)
```

data distribution

```
processors  P(NUMBER_OF_PROCESSORS)
distribute(*,BLOCK) onto P :: A, B
```

communication
compiler-generated

K. Kennedy, C. Koelbel, and H. Zima: The Rise and Fall of High Performance Fortran: An Historical Object Lesson

Proc. History of Programming Languages III (HOPL III), San Diego, June 2007

Fortran+MPI Communication for 3D 27-point Stencil (NAS MG rprj3)

```
subroutine comm3(u,n1,n2,n3,kk)
use caf_intrinsics

implicit none

include 'cafnpb.h'
include 'globals.h'

integer n1, n2, n3, kk
double precision u(n1,n2,n3)
integer axis

if( .not. dead(kk) )then
   do  axis = 1, 3
      if( nprocs .ne. 1) then
         call sync_all()
         call give3( axis, +1, u, n1, n2, n3, kk )
         call give3( axis, -1, u, n1, n2, n3, kk )
         call sync_all()
         call take3( axis, -1, u, n1, n2, n3 )
         call take3( axis, +1, u, n1, n2, n3 )
      else
         call comm1p( axis, u, n1, n2, n3, kk )
      endif
   enddo
else
   do  axis = 1, 3
      call sync_all()
      call sync_all()
   enddo
   call zero3(u,n1,n2,n3)
endif
return
end


subroutine give3( axis, dir, u, n1, n2, n3, k )
use caf_intrinsics


implicit none

include 'cafnpb.h'
include 'globals.h'

integer axis, dir, n1, n2, n3, k, ierr
double precision u( n1, n2, n3 )

integer i3, i2, i1, buff_len,buff_id

buff_id = 2 + dir
buff_len = 0

if( axis .eq.  1 )then
   if( dir .eq. -1 )then

      do  i3=2,n3-1
         do  i2=2,n2-1
            buff_len = buff_len + 1
            buff(buff_len,buff_id ) = u( 2,  i2,i3)
         enddo
      enddo

      buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>     buff(1:buff_len,buff_id)

   else if( dir .eq. +1 ) then

      do  i3=2,n3-1
         do  i2=2,n2-1
            buff_len = buff_len + 1
            buff(buff_len, buff_id ) = u( n1-1, i2,i3)
         enddo
      enddo

      buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>     buff(1:buff_len,buff_id)

   endif
endif

if( axis .eq.  2 )then
   if( dir .eq. -1 )then

      do  i3=2,n3-1
         do  i1=1,n1
            buff_len = buff_len + 1
            buff(buff_len, buff_id ) = u( i1,  2,i3)
         enddo
      enddo

      buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>     buff(1:buff_len,buff_id)

   else if( dir .eq. +1 ) then

      do  i3=2,n3-1
         do  i1=1,n1
            buff_len = buff_len + 1
            buff(buff_len,  buff_id )= u( i1,n2-1,i3)
         enddo
      enddo

      buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>     buff(1:buff_len,buff_id)

   endif
endif

if( axis .eq.  3 )then
   if( dir .eq. -1 )then

      do  i2=1,n2
         do  i1=1,n1
            buff_len = buff_len + 1
            buff(buff_len, buff_id ) = u( i1,i2,2)
         enddo
      enddo

      buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>     buff(1:buff_len,buff_id)

   else if( dir .eq. +1 ) then

      do  i2=1,n2
         do  i1=1,n1
            buff_len = buff_len + 1
            buff(buff_len, buff_id ) = u( i1,i2,n3-1)
         enddo
      enddo

      buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>     buff(1:buff_len,buff_id)

   endif
endif

return
end


subroutine take3( axis, dir, u, n1, n2, n3 )
use caf_intrinsics


implicit none

include 'cafnpb.h'
include 'globals.h'

integer axis, dir, n1, n2, n3
double precision u( n1, n2, n3 )

integer buff_id, indx

integer i3, i2, i1

buff_id = 3 + dir
indx = 0

if( axis .eq.  1 )then
   if( dir .eq. -1 )then

      do  i3=2,n3-1
         do  i2=2,n2-1
            indx = indx + 1
            u(n1,i2,i3) = buff(indx, buff_id )
         enddo
      enddo

   else if( dir .eq. +1 ) then

      do  i3=2,n3-1
         do  i2=2,n2-1
            indx = indx + 1
            u(1,i2,i3) = buff(indx, buff_id )
         enddo
      enddo

   endif
endif

if( axis .eq.  2 )then
   if( dir .eq. -1 )then

      do  i3=2,n3-1
         do  i1=1,n1
            indx = indx + 1
            u(i1,n2,i3) = buff(indx, buff_id )
         enddo
      enddo

   else if( dir .eq. +1 ) then

      do  i3=2,n3-1
         do  i1=1,n1
            indx = indx + 1
            u(i1,1,i3) = buff(indx, buff_id )
         enddo
      enddo

   endif
endif

if( axis .eq.  3 )then
   if( dir .eq. -1 )then

      do  i2=1,n2
         do  i1=1,n1
            indx = indx + 1
            u(i1,i2,n3) = buff(indx, buff_id )
         enddo
      enddo

   else if( dir .eq. +1 ) then

      do  i2=1,n2
         do  i1=1,n1
            indx = indx + 1
            u(i1,i2,1) = buff(indx, buff_id )
         enddo
      enddo

   endif
endif

return
end


subroutine comm1p( axis, u, n1, n2, n3, kk )
use caf_intrinsics


implicit none

include 'cafnpb.h'
include 'globals.h'

integer axis, dir, n1, n2, n3
double precision u( n1, n2, n3 )

integer i3, i2, i1, buff_len,buff_id
integer i, kk, indx

dir = -1

buff_id = 3 + dir
buff_len = nm2

do  i=1,nm2
   buff(i,buff_id) = 0.0D0
enddo


dir = +1

buff_id = 3 + dir
buff_len = nm2

do  i=1,nm2
   buff(i,buff_id) = 0.0D0
enddo

dir = +1

buff_id = 2 + dir
buff_len = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         buff_len = buff_len + 1
         buff(buff_len, buff_id ) = u( n1-1, i2,i3)
      enddo
   enddo
endif

if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         buff_len = buff_len + 1
         buff(buff_len,  buff_id )= u( i1,n2-1,i3)
      enddo
   enddo
endif

if( axis .eq.  3 )then
   do  i2=1,n2
      do  i1=1,n1
         buff_len = buff_len + 1
         buff(buff_len, buff_id ) = u( i1,i2,n3-1)
      enddo
   enddo
endif

dir = -1

buff_id = 2 + dir
buff_len = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         buff_len = buff_len + 1
         buff(buff_len,buff_id ) = u( 2,  i2,i3)
      enddo
   enddo
endif

if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         buff_len = buff_len + 1
         buff(buff_len, buff_id ) = u( i1,  2,i3)
      enddo
   enddo
endif

if( axis .eq.  3 )then
   do  i2=1,n2
      do  i1=1,n1
         buff_len = buff_len + 1
         buff(buff_len, buff_id ) = u( i1,i2,2)
      enddo
   enddo
endif

do  i=1,nm2
   buff(i,4) = buff(i,3)
   buff(i,2) = buff(i,1)
enddo

dir = -1

buff_id = 3 + dir
indx = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         indx = indx + 1
         u(n1,i2,i3) = buff(indx, buff_id )
      enddo
   enddo
endif

if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         indx = indx + 1
         u(i1,n2,i3) = buff(indx, buff_id )
      enddo
   enddo
endif

if( axis .eq.  3 )then
   do  i2=1,n2
      do  i1=1,n1
         indx = indx + 1
         u(i1,i2,n3) = buff(indx, buff_id )
      enddo
   enddo
endif


dir = +1

buff_id = 3 + dir
indx = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         indx = indx + 1
         u(1,i2,i3) = buff(indx, buff_id )
      enddo
   enddo
endif

if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         indx = indx + 1
         u(i1,1,i3) = buff(indx, buff_id )
      enddo
   enddo
endif

if( axis .eq.  3 )then
   do  i2=1,n2
      do  i1=1,n1
         indx = indx + 1
         u(i1,i2,1) = buff(indx, buff_id )
      enddo
   enddo
endif

return
end
```

Chapel 3D NAS MG Stencil rprj3

```
function rprj3(S,R) {
  const Stencil: domain(3) = [-1..1, -1..1, -1..1],        // 27-points
        w: [0..3]real = (/0.5, 0.25, 0.125, 0.0625/),       // weights
        w3d: [(i,j,k) in Stencil] = w((i!=0) + (j!=0) + (k!=0));

forall ijk in S.domain do
    S(ijk) = sum reduce [off in Stencil] (w3d(off) * R(ijk + R.stride*off));
}
```

Productivity Challenges for Peta-Scale Systems

- **Large-scale hierarchical architectural parallelism**
  - *tens of thousands to hundreds of thousands of processors*
  - *component failures may occur frequently*
- **Extreme non-uniformity in data access**
- **Applications: large, complex, and long-lived**
  - *multi-disciplinary, multi-language, multi-paradigm*
  - *dynamic, irregular, and adaptive*
  - *survive many hardware generations ➔ portability is important*
- **How to exploit the parallelism and locality provided by the architecture?**
  - *automatic parallelization and locality management are not powerful enough to provide a general efficient solution*
  - *explicit support for control of parallelism and locality must be provided by the programming model and the language*

Contents

Languages for High Performance Computing

- HPF Language Family
  - *predecessors: CM-Fortran, Fortran D, Vienna Fortran*
  - *High Performance Fortran (HPF): HPF-1 (1993); HPF-2(1997)*
  - *successors: HPF+, HPF/JA*
- OpenMP
- Partitioned Global Address Space (PGAS) Languages
  - *Co-Array Fortran*
  - *UPC*
  - *Titanium*
- High-Productivity Languages developed in the HPCS Program
  - *Chapel*
  - *X10*
  - *Fortress*
- Domain-Specific Languages and Abstractions

PGAS Language Overview

Support for global view of data, but local control

- **Partitioned Global Address Space (PGAS) languages are based on the Single-Program-Multiple-Data (SPMD) model**
- **Providing a shared-memory, *global view,* of data, combined with support for locality**
  - ***global address space is logically partitioned, mapped to processors***
  - ***single-sided shared-memory communication***
  - ***local and remote references distinguished in the source code***
  - ***implemented via one-sided communication libraries (e.g., GASNet)***
- ***Local control* of execution via processor-centric view**
- **Main representatives: *Co-Array Fortran (CAF), Unified Parallel C (UPC), Titanium***

Example: PGAS vs. HPCS

Setting up a block-distributed array in Titanium vs. Chapel

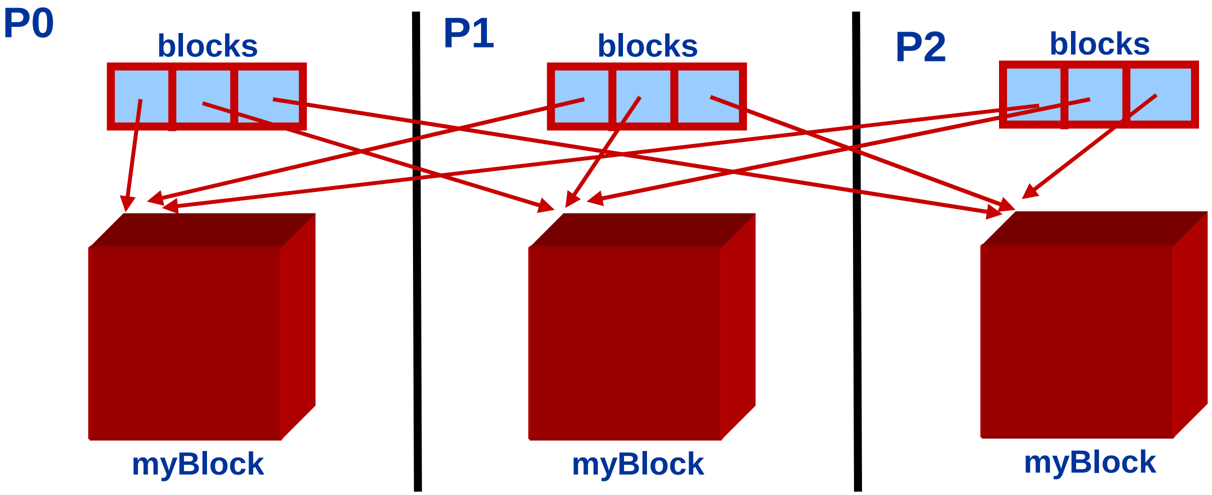
Titanium: *a dialect of Java that supports distributed multi-dimensional arrays, iterators, subarrays, and synchronization/communication primitives*

Titanium Code Fragment

```
// determine parameters of local block:
Point<3> startCell = myBlockPos * numCellsPerBlockSide;
Point<3> endCell   = startCell + (numCellsPerBlockSide-[1,1,1]);

//create local myBlock array:
double [3d] myBlock = new double[startCell:endCell];

//build the distributed structure:
//declare blocks as 1D-array of references (one element per processor)
blocks.exchange(myBlock);
```

Chapel Code Fragment

```
const D: domain(3)  =  [l1..u1,l2..u2,l3..u3]
               distributed(block,block,block);
…
var A: [D] real;
…
```



Source: K.Yelick et al.: Parallel Languages and Compilers: Perspective from the Titanium Experience

HPCS Languages

global view of data, global control

- *High-Productivity Computing Systems (HPCS)* is a DARPA-sponsored program for the development of peta-scale architectures (2002-2010)
- HPCS Languages
  - *Chapel (Cascade Project, led by Cray Inc.)*
  - *X10 (PERCS Project, led by IBM)*
  - *[Fortress (HERO Project [until 2006], led by Sun Microsystems)]*
- These are new, memory-managed, object-oriented languages
  - *global view of data and computation ➔ generally no distinction between local and remote data access in the source code*
  - *support for explicit data and task parallelism*
  - *explicit locality management*
  - Chapel *is unique in that it provides user-defined data distributions*

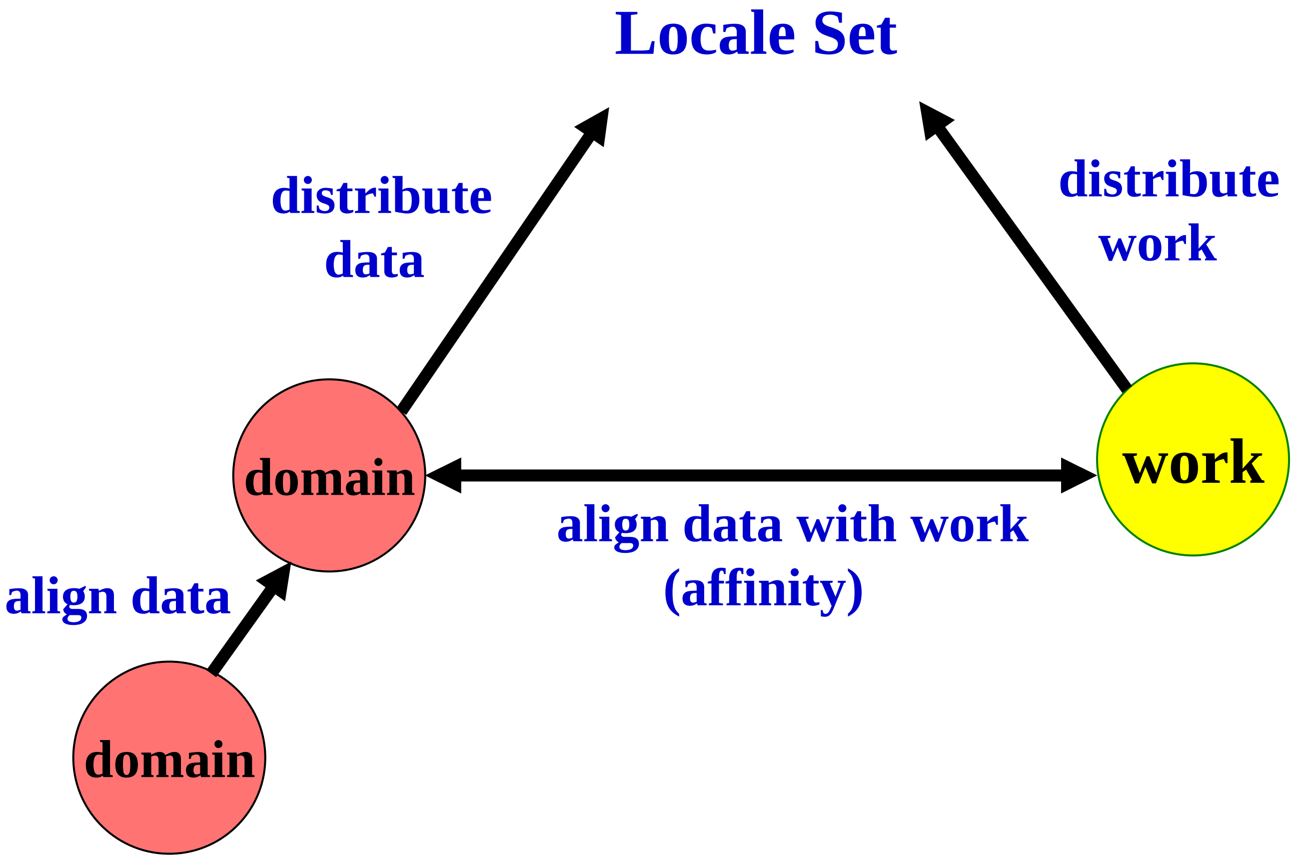
Chapel Language Concepts

http://chapel.cs.washington.edu

- Explicit high-level control of parallelism
  - *data parallelism*
    - domains, arrays, indices: *support distributed data aggregates*
    - forall loops and iterators: *express data parallel computations*
  - *task parallelism*
    - cobegin statements: *specify task parallel computations*
    - synchronization variables, atomic sections
- Explicit high-level control of locality
  - *"locales": abstract units of locality*
  - *data distributions: map data domains to sets of locales*
  - *on clauses: map execution components to sets of locales*
- Close relationship to mainstream languages
  - *object-oriented*
  - *modules for Programming-in-the-Large*

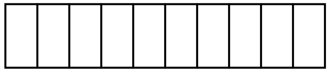
Aspects of Locality

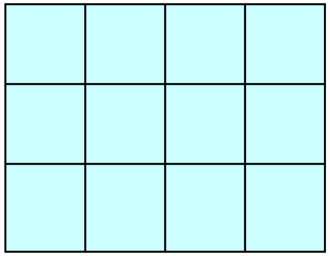
Locale: *an abstract unit of locality*



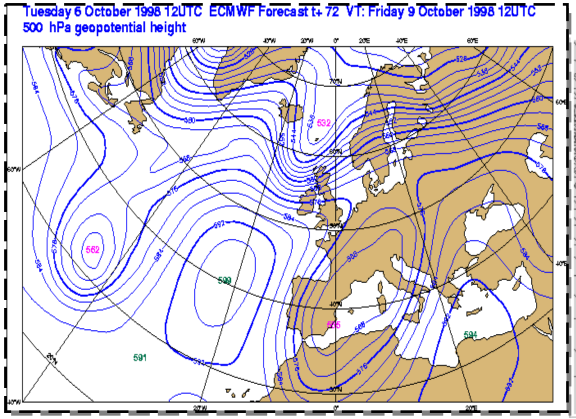
Data Distributions Can Be …

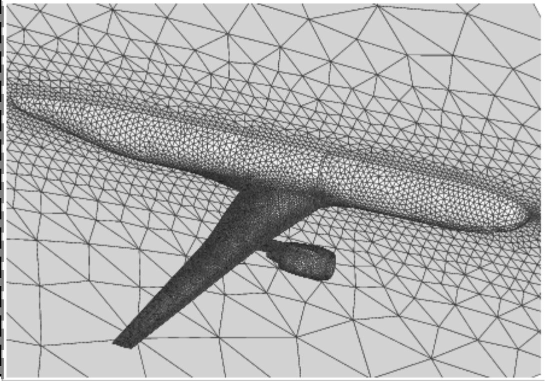
regular, and easy to deal with in the compiler/runtime system:

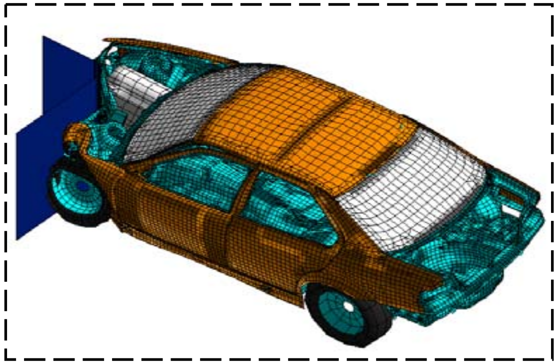


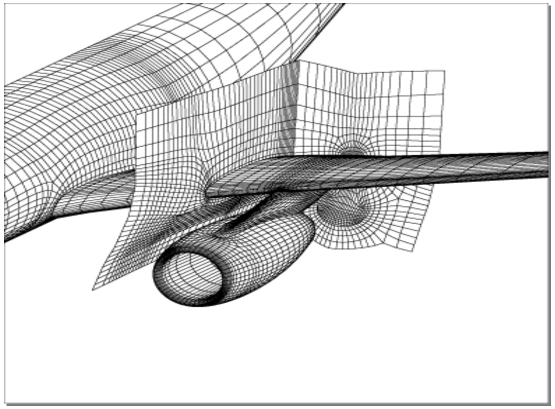


or irregular, possibly depending on runtime information:







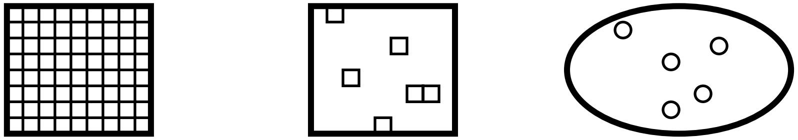


Domains

- Concept influenced by HPF templates, ZPL regions
- Domains are first-class objects
- Domain components
  - *index set*
  - *distribution*
  - *set of arrays*
- Index sets are general sets of “names”
  - *Cartesian products of integer intervals (as in Fortran95, etc.)*
  - *sparse subsets of Cartesian products*
  - *sets of object instances, e.g., for graph-based data structures*
- Iterators based on domains

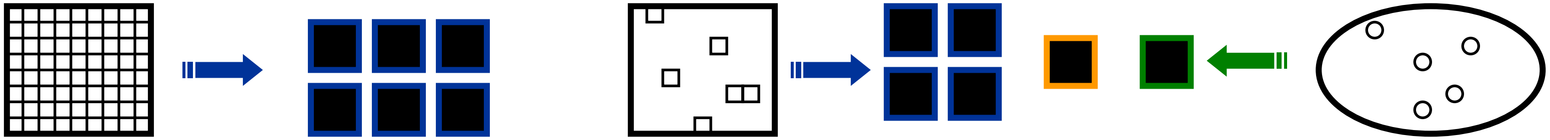
Domains and Distributions in Context

- *index sets: Cartesian products, sparse, sets*

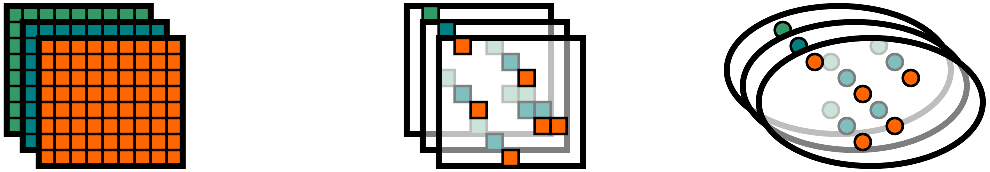


- *locale view*: a logical view for a set of locales

- *distribution*: a mapping of an index set to a locale view



- *array*: a map from an index set to a collection of variables



Source: Brad Chamberlain (Cray Inc.)

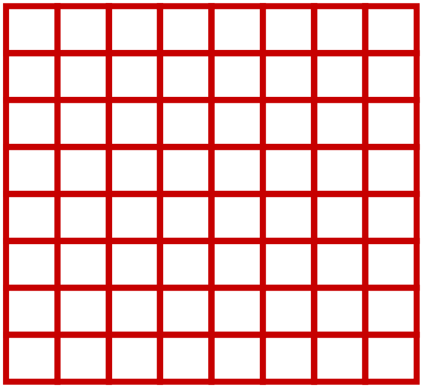
Example: Jacobi Relaxation in Chapel

```
const L:[1..p,1..q] locale = reshape(Locales);

const n= …, epsilon= …;
const DD:domain(2)=[0..n+1,0..n+1] distributed(block,block)on L;
      D: subdomain(DD) = [1..n, 1..n];
var delta: real;
var A, Temp: [DD] real; /*array declarations over domain DD */

A(0,1..n) = 1.0;

do {
    forall (i,j) in D {  /* parallel iteration over domain D */
       Temp(i,j) = (A(i-1,j)+A(i+1,j)+A(i,j-1)+A(i,j+1))/4.0;
       delta = max reduce abs(A(D) – Temp(D));
       A(D) = Temp(D);
    } while (delta > epsilon);

writeln(A);
```

Example: Jacobi Relaxation in Chapel

```
const L:[1..p,1..q] locale = reshape(Locales);

const n= …, epsilon= …;
const DD:domain(2)…distributed(block,block) on L;
      D: subdomain(DD) = [1..n, 1..n];
var delta: real;
var A, Temp: [DD] real;

A(0,1..n) = 1.0;

do {
    forall (i,j) in D {
       Temp(i,j) = (A(i-1,j)+A(i+1,j)+A(i,j-1)+A(i,j+1))/4.0;
       delta = max reduce abs(A(D) – Temp(D));
       A(D) = Temp(D);
    } while (delta > epsilon);

writeln(A);
```

Locale Grid L



Key Features

- •global view of data/control
- •explicit parallelism (forall)
- •high-level locality control
- •NO explicit communication
- •NO local/remote distinction in source code

Chapel’s Framework for User-Defined Distributions

◆ **Provides functionality for:**

- *distributing index sets across locales*
- *arranging data within a locale*
- *defining specialized distribution libraries*

◆ **This capability is in its effect similar to *function specification***

- *unstructured meshes*
- *multi-block problems*
- *multi-grid problems*
- *distributed sparse matrices*

Locality Control in Chapel: Basic Concepts

◆ **Domain: first class entity**

- *components: index set, distribution, associated arrays, iterators*

◆ **Array—Mapping from a Domain to a Set of Variables**

◆ **Framework for User-Defined Distributions: three levels**

1. *naïve use of a predefined library distribution (block, cyclic, indirect,…)*
2. *specification of a distribution by*

   global mapping: index set → locales

   - *interface for the definition of mapping, distribution segments, iterators*
   - *system-provided default functionality can be overridden by user*
3. *specification of a distribution by global mapping and*

   layout mapping: index set → locale data space

◆ **High-Level Control of *Communication***

- *user-defined specification of halos; communication assertions*

User-Defined Distributions: Global Mapping

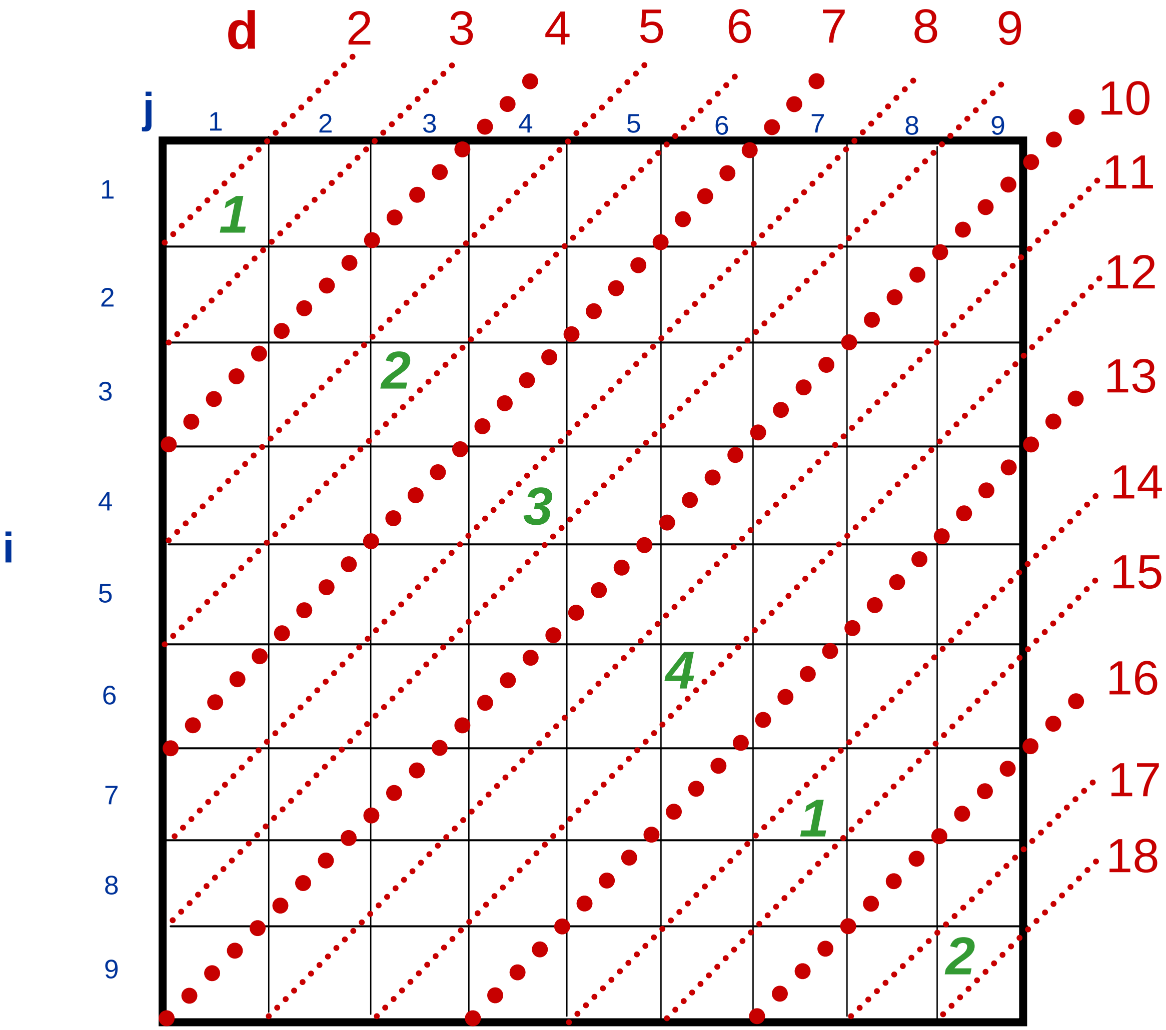
/* declaration of distribution classes MyC and MyB: */

```
class MyC: Distribution {
  const z:int;                                      /* block size */
  const ntl:int;                                    /* number of target locales*/

  function map(i:index(source)):locale {            /* global mapping for MyC */
    return Locales(mod(ceil(i/z-1)+1,ntl));
  }

class MyB: Distribution {
  var bl:int = ...;                                 /* block length */

  function map(i: index(source)):locale {           /* global mapping for MyB */
    return Locales(ceil(i/bl));
  }
}
```

/* use of distribution classes MyC and MyB in declarations: */

```
const D1C: domain(1) distributed(MyC(z=100))=1..n1;
const D1B: domain(1) distributed(MyB) on Locales(1..num_locales/10)=1..n1;
var   A1: [D1C] real;
var   A2: [D1B] real;
```

Example: Banded Distribution



Diagonal A/d = { A(i,j) | d=i+j }

bw = 3 (bandwidth)

p=4 (number of locales)

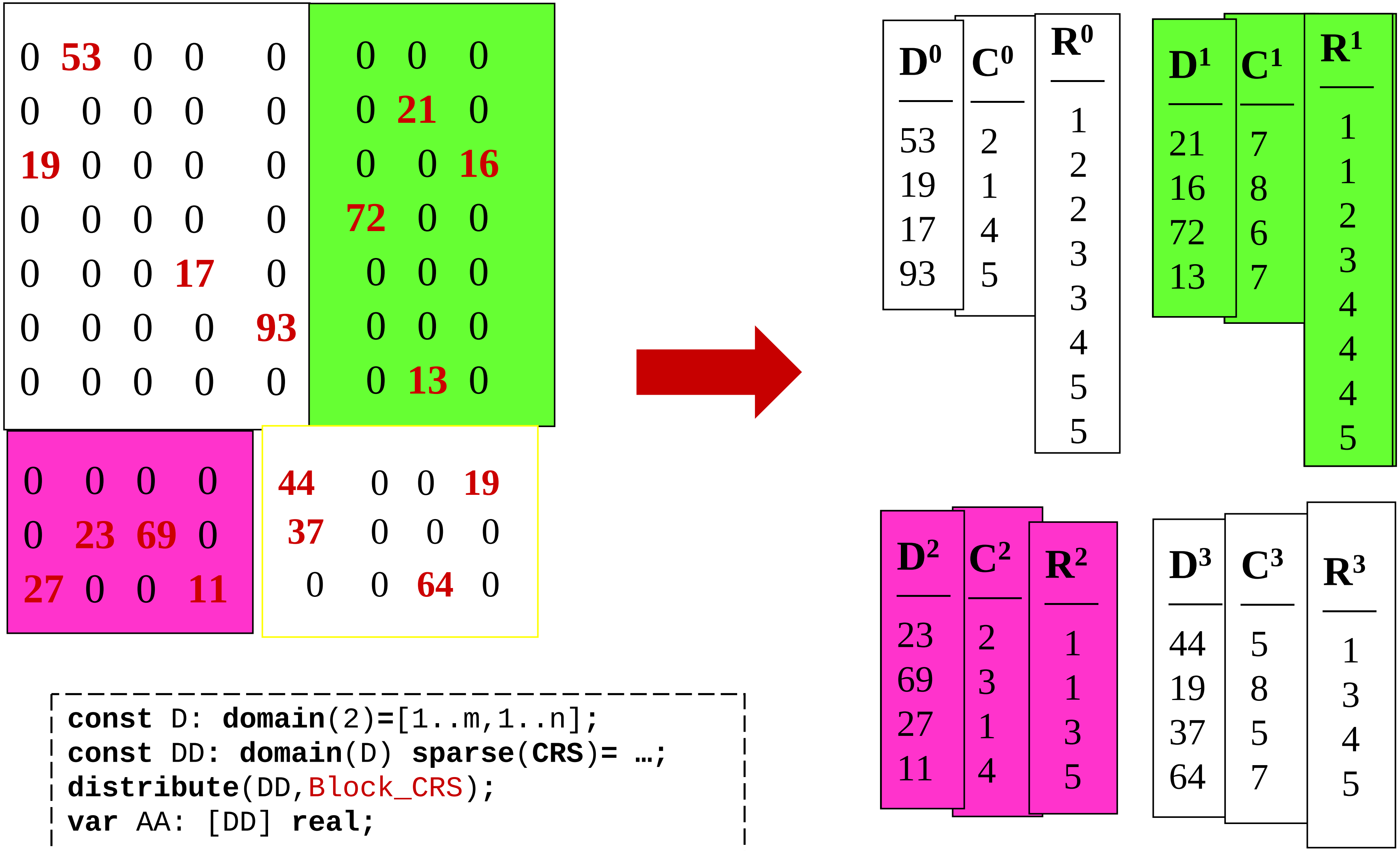
Distribution—global map:

Blocks of bw diagonals are cyclically mapped to locales

Layout:

Each diagonal is represented as a one-dimensional dense array. Arrays in a locale are referenced by a pointer array

Example
Matrix-Vector Multiplication (sparse CRS)



0	**53**	0	0	0	0	0	0
0	0	0	0	0	0	**21**	0
19	0	0	0	0	0	0	**16**
0	0	0	0	0	**72**	0	0
0	0	0	**17**	0	0	0	0
0	0	0	0	**93**	0	0	0
0	0	0	0	0	0	**13**	0
0	0	0	0	**44**	0	0	**19**
0	**23**	**69**	0	**37**	0	0	0
27	0	0	**11**	0	0	**64**	0

D^0	C^0	R^0
53	2	1
19	1	2
17	4	2
93	5	3
		3
		4
		5
		5

D^1	C^1	R^1
21	7	1
16	8	1
72	6	2
13	7	3
		4
		4
		4
		5

D^2	C^2	R^2
23	2	1
69	3	1
27	1	3
11	4	5

D^3	C^3	R^3
44	5	1
19	8	3
37	5	4
64	7	5

```
const D: domain(2)=[1..m,1..n];
const DD: domain(D) sparse(CRS)= …;
distribute(DD,Block_CRS);
var AA: [DD] real;

…
```

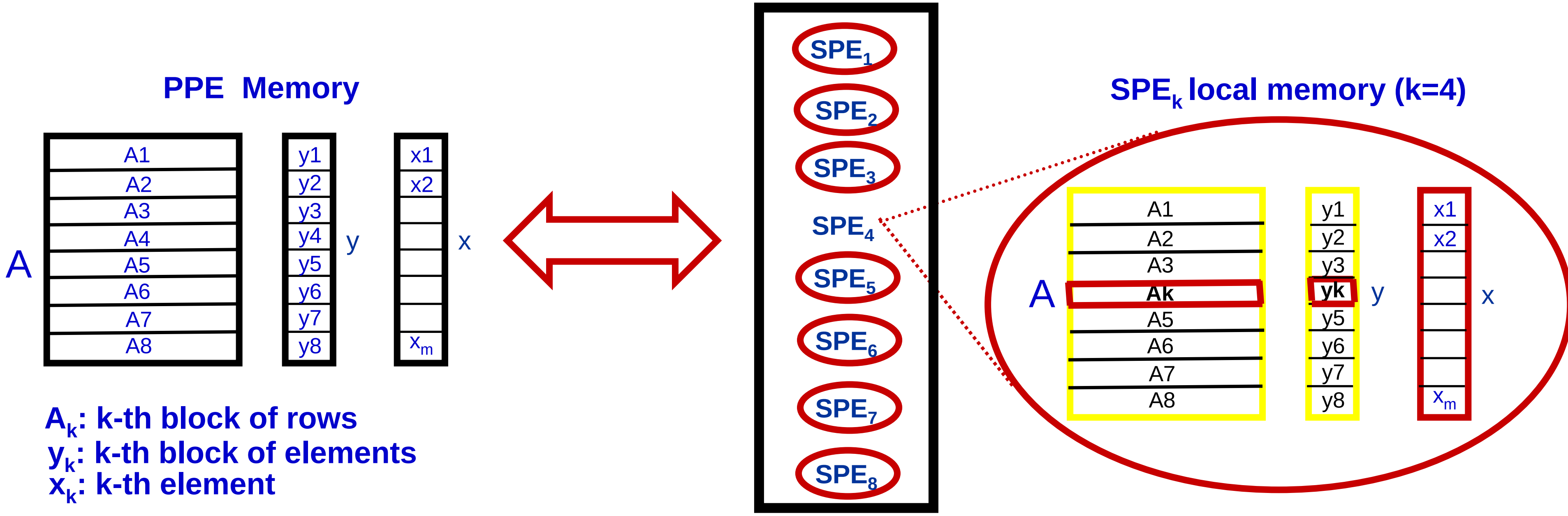
Example: Heterogeneous Distributions Matrix-Vector Multiply on the Cell

```
var A: [1..m,1..n] real;
var x: [1..n]      real;
var y: [1..m]      real;

y = sum reduce(dim=2) forall (i,j) in [1..m,1..n] A(i,j)*x(j);
```

(original) Chapel version

```
param n_spe = 8;                   /* number of synergistic processors (SPEs) */
const SPE:[1..n_spe] locale;       /* declaration of SPE array */

var A: [1..m,1..n] real distributed(block,*) on SPE;
var x: [1..n]      real replicated           on SPE;
var y: [1..m]      real distributed(block)   on SPE;

y = sum reduce(dim=2) forall (i,j) in [1..m,1..n] A(i,j)*x(j);
```

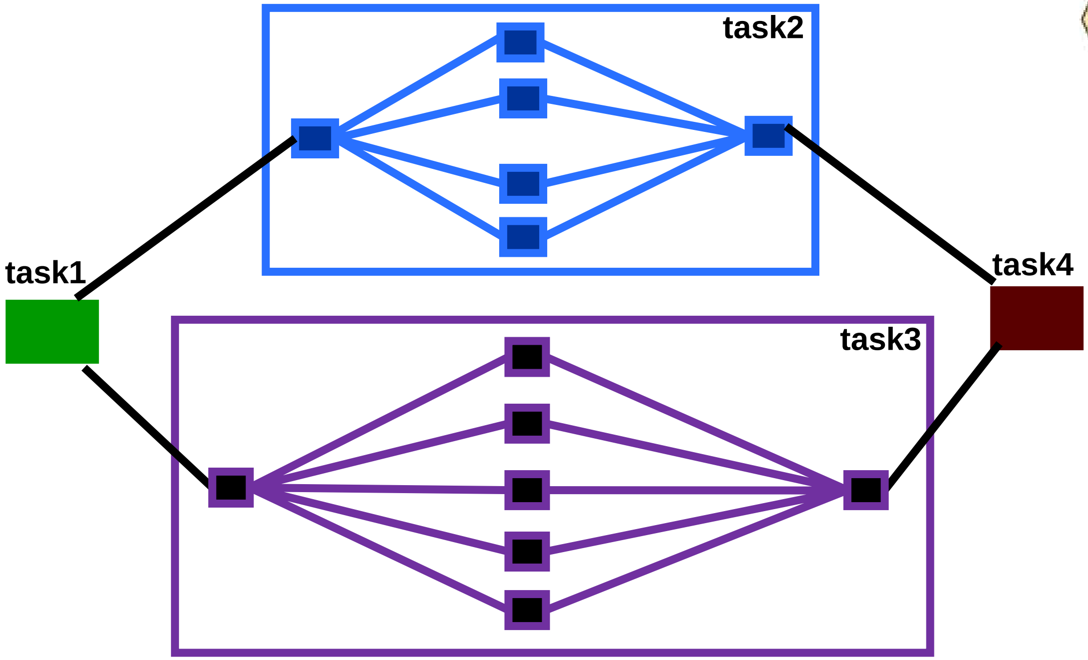
Chapel with (implicit) heterogeneous semantics

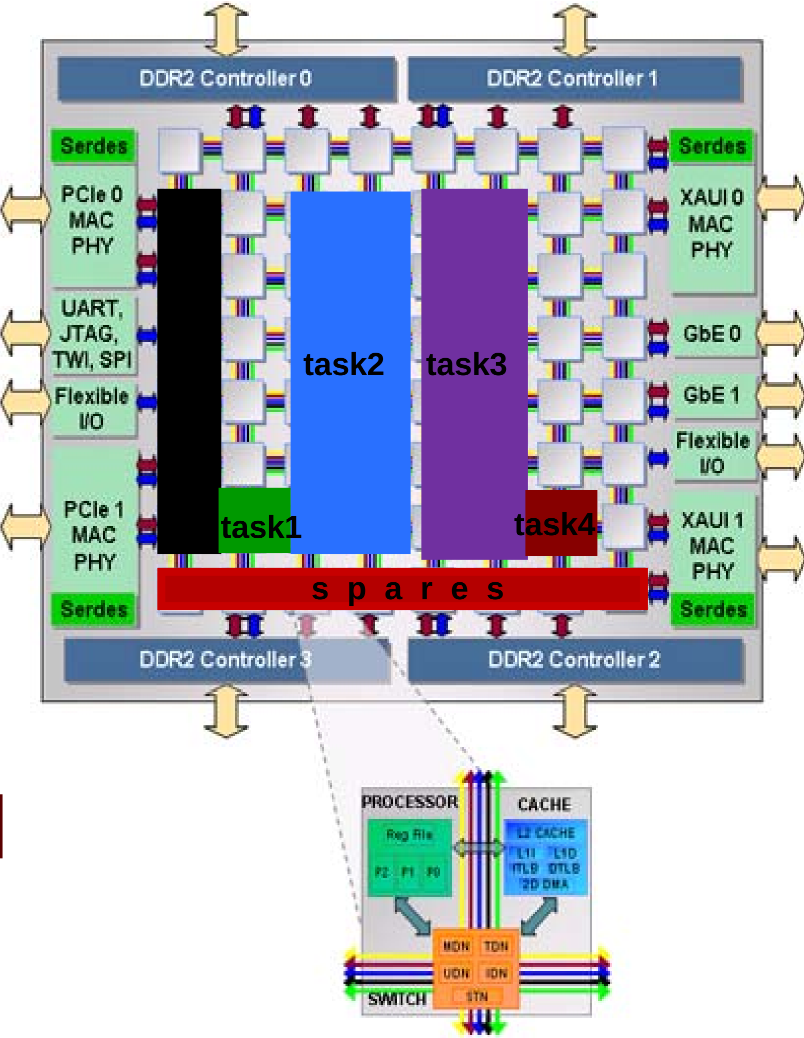


A_k: k-th block of rows
y_k: k-th block of elements
x_k: k-th element

Example: Nested Task and Data Parallelism

```
! In task2:
var A:[m1,m2]float distributed(…)on …;
  …
forall (i,j) in A do …

! In task3:
var B:[m]… distributed(…)on …;
  …
forall k in B do …
```





Contents

Compiler/Runtime Technology for High-Level Locality Management

- **Suprenum Project (Bonn University)**

  First translator

  Fortran 77 + data distribution spec→ Message Passing Fortran

  (Michael Gerndt's Ph.D. work, 1989)

- **Compilation/Runtime Technology for irregular distributions developed in the context of Fortran D, Vienna Fortran, HPF-2, and other approaches in the 1990s**

- **Architecture/Application Adaptive Compilation and Runtime Technology**

- **Introspection Technology**

Inspector/Executor Method

(Koelbel, Mehrotra, Saltz)

```
forall i in D on home(c(k(i))) independent {
   y(k(i)) = x(i) + c(k(i)) * z(k(i))
}
```

Generated code for processor *p*

INSPECTOR:

Loop analysis: *determine iteration sets and for all p’ all sets RCV(p,p’) of data elements owned by p’ and accessed in p*

Compute send sets: *SENDS(p.p’) of data elements that need to be sent from p to p’ for all p’*

EXECUTOR:

Send: *for all p’ such that SENDS(p.p’) is non-empty send all data in SENDS(p,p’) to p’*

Execute local iterations

Receive: *for all p’ such that RCV(p,p’) is non-empty receive data in RCV(p,p’) into a local TEMP*

Execute non-local iterations locally

Architecture- and Application-Adaptive Compilation and Runtime Technology

- **Code generation technology inspired by ATLAS and similar systems**

- **Hybrid approach**
  - *model-guided: static models of architecture, profitability*
    - *these are the conventional methods of compiler analysis*
    - *for theoretical and practical reasons results are in general sub-optimal*
  - *empirical optimization using actual execution of parameterized code, intelligent search*

- **Exploit complementary strengths of both methods:**
  - *static compiler technology reduces search space by pruning unprofitable solutions*
  - *empirical data provide accurate measure of optimization impact*

Note: Our HPDC conference paper describes this approach in detail

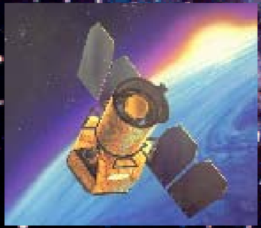
Contents

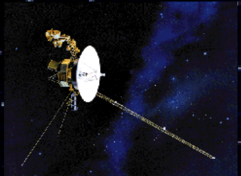
High Performance Computing and Embedded Computing: Common Issues

- **High Performance Computing (HPC) and Embedded Computing (EC) have been traditionally at the extremes of the computational spectrum**
- **However, future HPC, EC, and HPEC systems will need to address many similar issues (at different scales):**
  - ***multi-core as the underlying technology***
  - ***massive parallelism at multiple levels***
  - ***power consumption constraints***
  - ***fault tolerance***
  - ***high-productivity reusable software***

More than 50 NASA Missions Explore Our Solar System

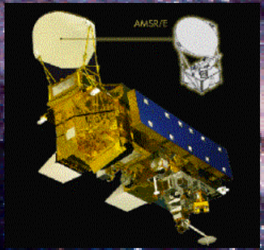

Spitzer studying stars and galaxies in the infrared


GALEX surveying galaxies in the ultraviolet

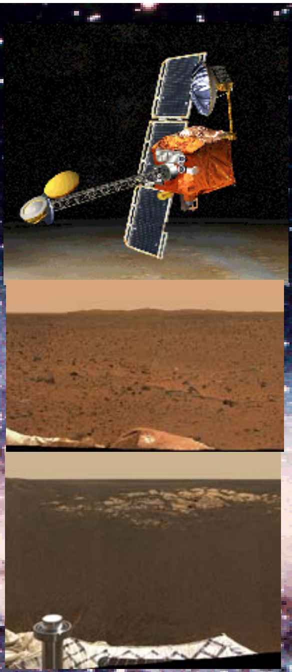

Two Voyagers on an interstellar mission


Chandra studying the x-ray universe


Ulysses studying the sun


Aqua studying Earth's oceans


Aura studying Earth's atmosphere


Mars Odyssey, rovers "Spirit" and "Opportunity" studying Mars


Hubble studying the universe


Cassini studying Saturn


CALIPSO studying Earth's climate


MESSENGER on its way to Mercury


New Horizons on its way to Pluto


QuikScat, Jason 1, CloudSat, and GRACE (plus ASTER, MISR, AIRS, MLS and TES instruments) monitoring Earth.

Space Challenges: Environment

Constraints on Spacecraft Hardware

- **Radiation**
  - *Total Ionizing Dose (TID)—amount of ionizing radiation over time: can lead to long-term cumulative degradation, permanent damage*
  - *Single Event Effects—caused by a single high-energy particle traveling through a semiconductor and leaving a ionized trail*
    - *Single Event Latchup (SEL)—catastrophic failure of the device (prevented by Silicon-On-Insulator (SOI) technology)*
    - *Single Event Upset (SEU) and Multiple Bit Upset (MBU)—change of bits in memory: a transient effect, causing no lasting damage*
- **Temperature**
  - *wide range (from -170 C on Europa to >400 C on Venus)*
  - *short cycles (about 50 C on MER)*
- **Vibration**
  - *launch*
  - *Planetary Entry, Descent, Landing (EDL)*

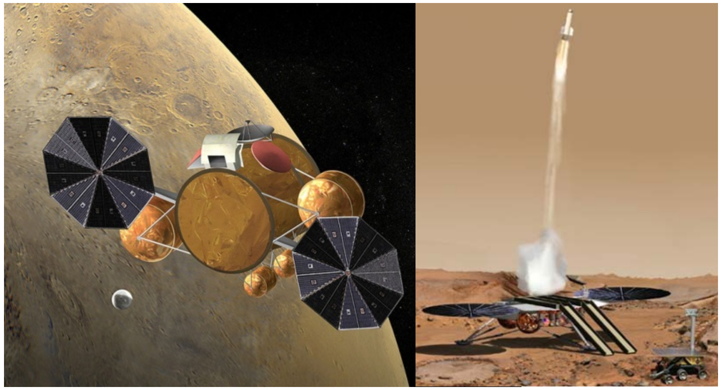
Space Challenges: Communication and Navigation

Constraints on mission operations

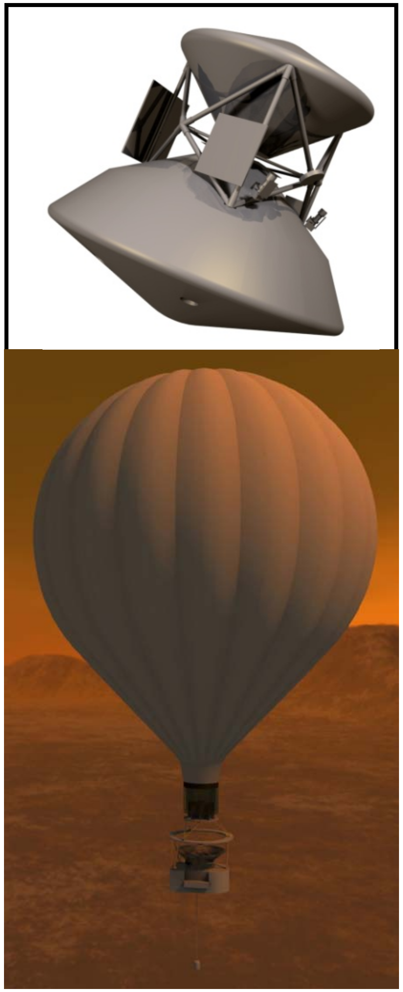
- **Bandwidth**
  - ***6 Mbit/s maximum, but typically much less (100 b/s)***
  - ***spacecraft transmitter power less than light bulb in a refrigerator***
- **Latency (one way)**
  - ***20 minutes to Mars***
  - ***13 hours to Voyager 1***
- **Navigation**
  - ***Position***
  - ***Velocity***

NASA/JPL: Potential Future Missions

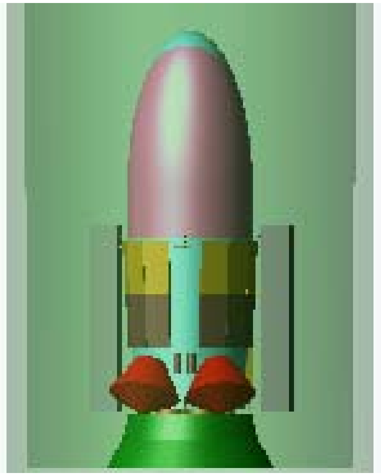
Artist Concept



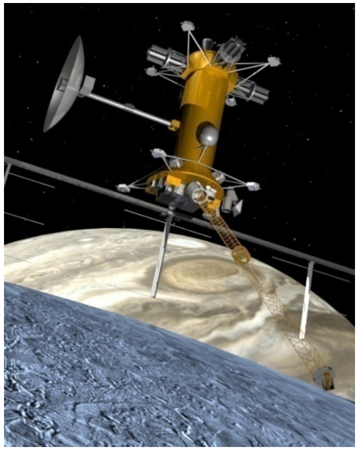
Mars Sample Return



Titan Explorer



Neptune Triton Explorer



Europa Explorer

Europa Astrobiology Laboratory

New Requirements

New applications and the limited downlink to Earth lead to two major new requirements:

1. **Autonomy**
2. **High-Capability On-Board Computing**

Such missions require on-board computational power ranging from tens of Gigaflops to hundreds of Teraflops. Emerging multi-core technology provides this capability.

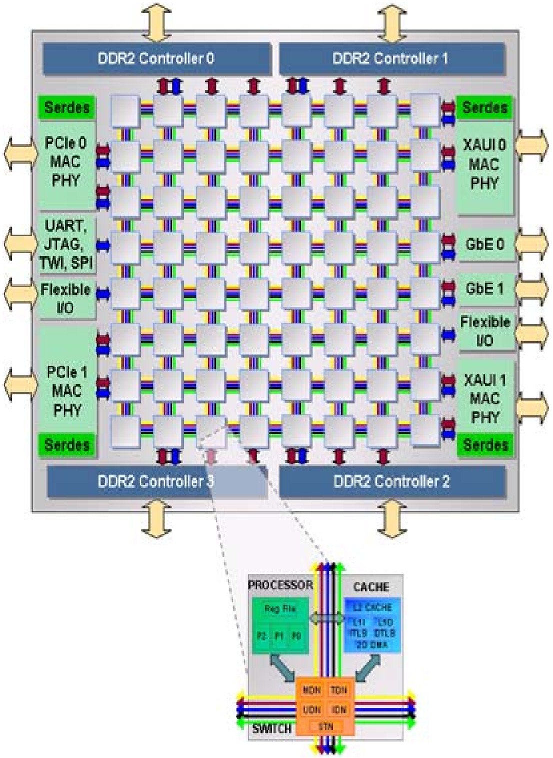
The Traditional Approach will not Scale

- **The traditional approach to space-borne computing is based on radiation-hardened processors and fixed redundancy (e.g.,Triple Modular Redundancy—TMR)**
  - ***Current Generation (Phoenix and Mars Science Lab –'09 Launch)***
    - ***Single BAE Rad 750 Processor***
    - ***256 MB of DRAM and 2 GB Flash Memory (MSL)***
    - ***200 MIPS peak, 14 Watts available power (14 MIPS/W)***

- **Radiation-hardened processors today lag commercial architectures by a factor of up to 100**

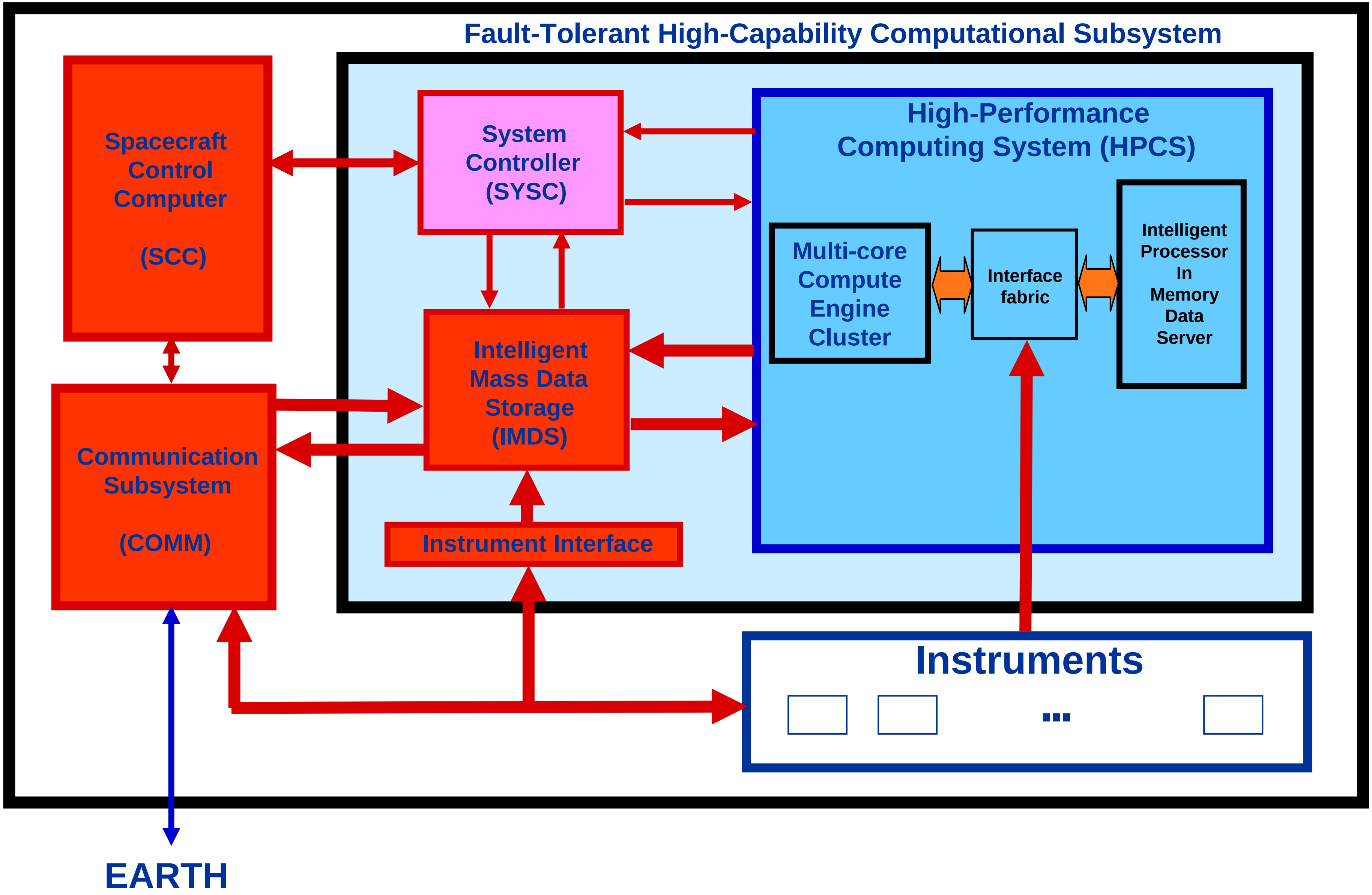
Multi-Core Systems
Will Provide the Required Capability

- **Tile64 (Tilera Corporation, 2007)**
  - *64 identical cores, arranged in an 8X8 grid*
  - *iMesh on-chip network, 27 Tb/sec bandwidth*
  - *170-300mW per core; 600 MHz – 1 GHz*
  - *192 GOPS (32 bit)—about 10 GOPS/Watt*



- **Maestro: a radiation-hardened version of Tile64 (announced for 2011)**
  - *currently in development at Boeing Corporation*
  - *49 cores, arranged in a 7X7 grid*
  - *70 GOPS at max power of 28W*

High-Capability On-Board System: A Hybrid Approach



Transient Faults

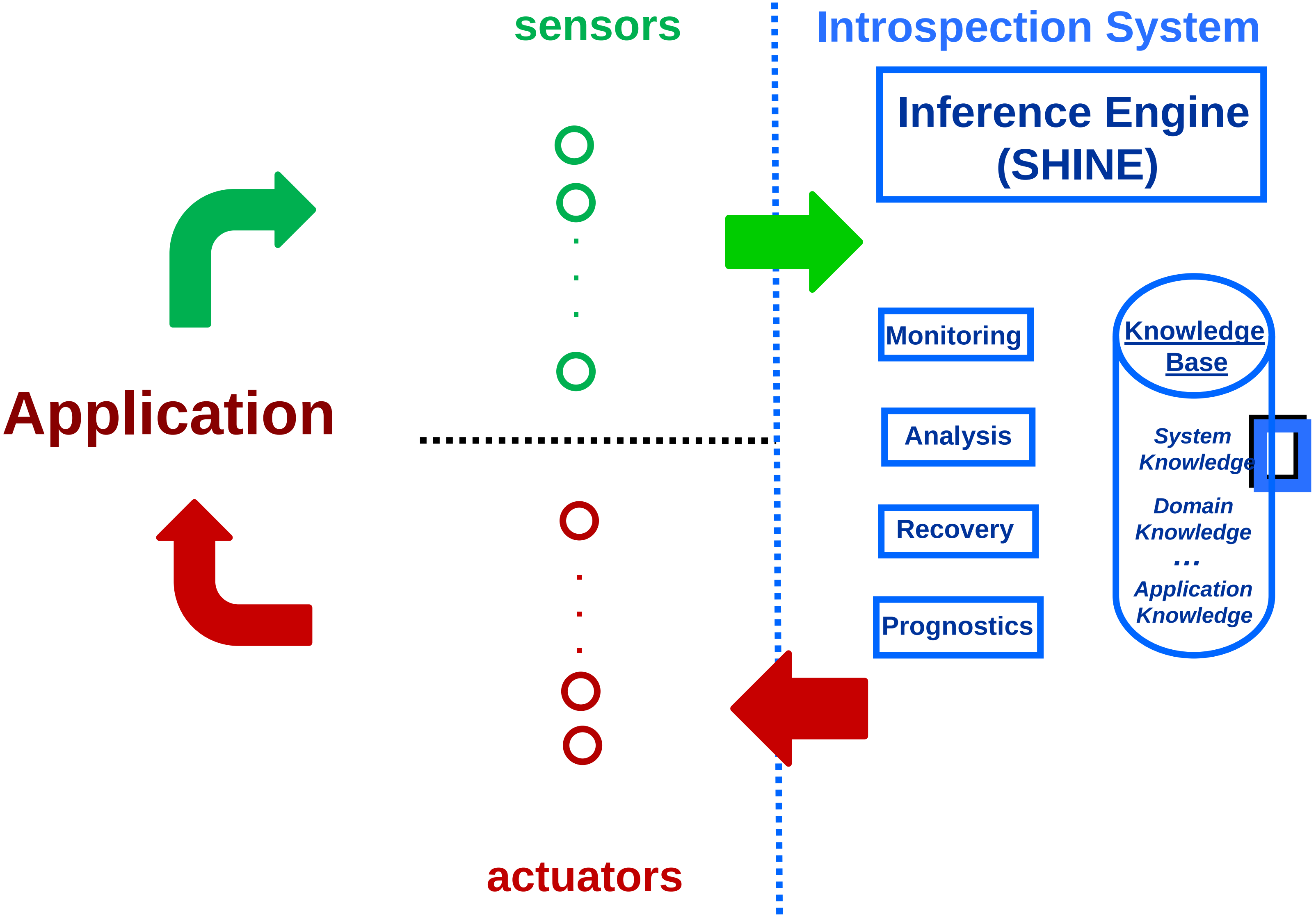
- **SEUs and MBUs are radiation-induced transient hardware errors, which may corrupt software in multiple ways:**
  - ***instruction codes and addresses***
  - ***user data structures***
  - ***synchronization objects***
  - ***protected OS data structures***
  - ***synchronization and communication***
- **Potential effects include:**
  - ***wrong or illegal instruction codes and addresses***
  - ***wrong user data in registers, cache, or DRAM***
  - ***control flow errors***
  - ***unwarranted exceptions***
  - ***hangs and crashes***
  - ***synchronization and communication faults***

Introspection…

- **provides *dynamic* monitoring, analysis, and feedback, enabling system to become self-aware and context-aware:**
    - ***monitoring execution behavior***
    - ***reasoning about its internal state***
    - ***changing the system or system state when necessary***
- **exploits adaptively the available threads**
- **can be applied to different scenarios, including:**
    - ***fault tolerance***
    - ***performance tuning***
    - ***power management***
    - ***behavior analysis***

This makes introspection technology applicable to on-board computing as well as to large-scale supercomputing

An Introspection Module (IM)



sensors

Introspection System

Inference Engine (SHINE)

Application

Monitoring

Analysis

Recovery

Prognostics

Knowledge Base

System Knowledge

Domain Knowledge

...

Application Knowledge

actuators

Conclusion

- Focus of this talk was on high-productivity *general-purpose languages*
  - *data parallelism—regular or irregular—is the main source of scalable parallelism*
  - *successful, industrial-strength implementations still under development*
- Research challenges remain
  - *performance porting of legacy applications*
  - *integration of codes in a multi-language-multi-paradigm environment*
  - *architecture- and application-adaptive compiler/runtime technology*
  - *intelligent tools for performance tuning, fault tolerance, power management*
- Domain-specific approaches represent viable high-level alternatives
- Heterogeneous systems and thread/task parallelism
  - *many approaches exist, almost all at a low level*
  - *explicit thread parallelism unmanageable for average programmer (E. Lee)*
  - *abstractions needed that concisely express typical patterns reliably*