Parallel Computing on Heterogeneous Networks

Alexey Lastovetsky

Department of Computer Science
University College Dublin
Alexey.Lastovetsky@ucd.ie

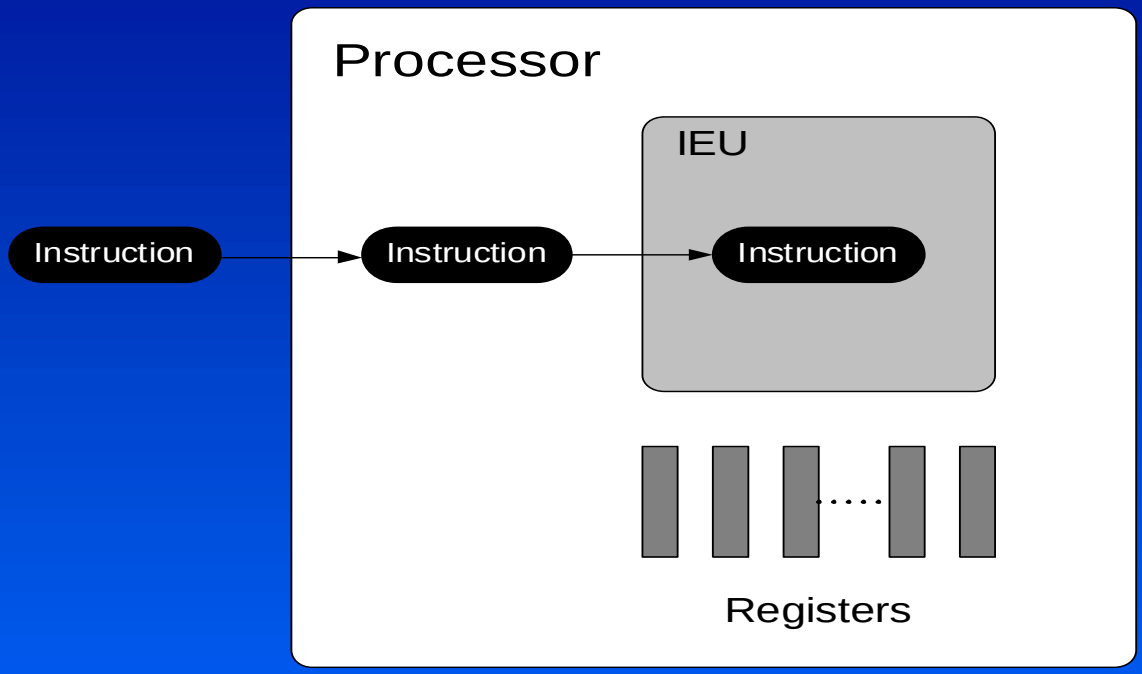
Parallel computing vs distributed comuting

- *Distributed computing makes* different *software* components inherently located on different computers *work together*
- *Parallel computing speeds up solving the problem* on the available computing resources. Distribution of computations over computers is just a way to speedup the program not its inherent property.

Evolution of parallel computing

- Evolution of parallel computing follows the evolution of computer hardware
- Main architecture milestones
  - Serial scalar processor
  - Vector and superscalar processors
  - Shared-memory multiprocessor (SMP)
  - Distributed-memory multiprocessor (MPP)
  - Network of computers

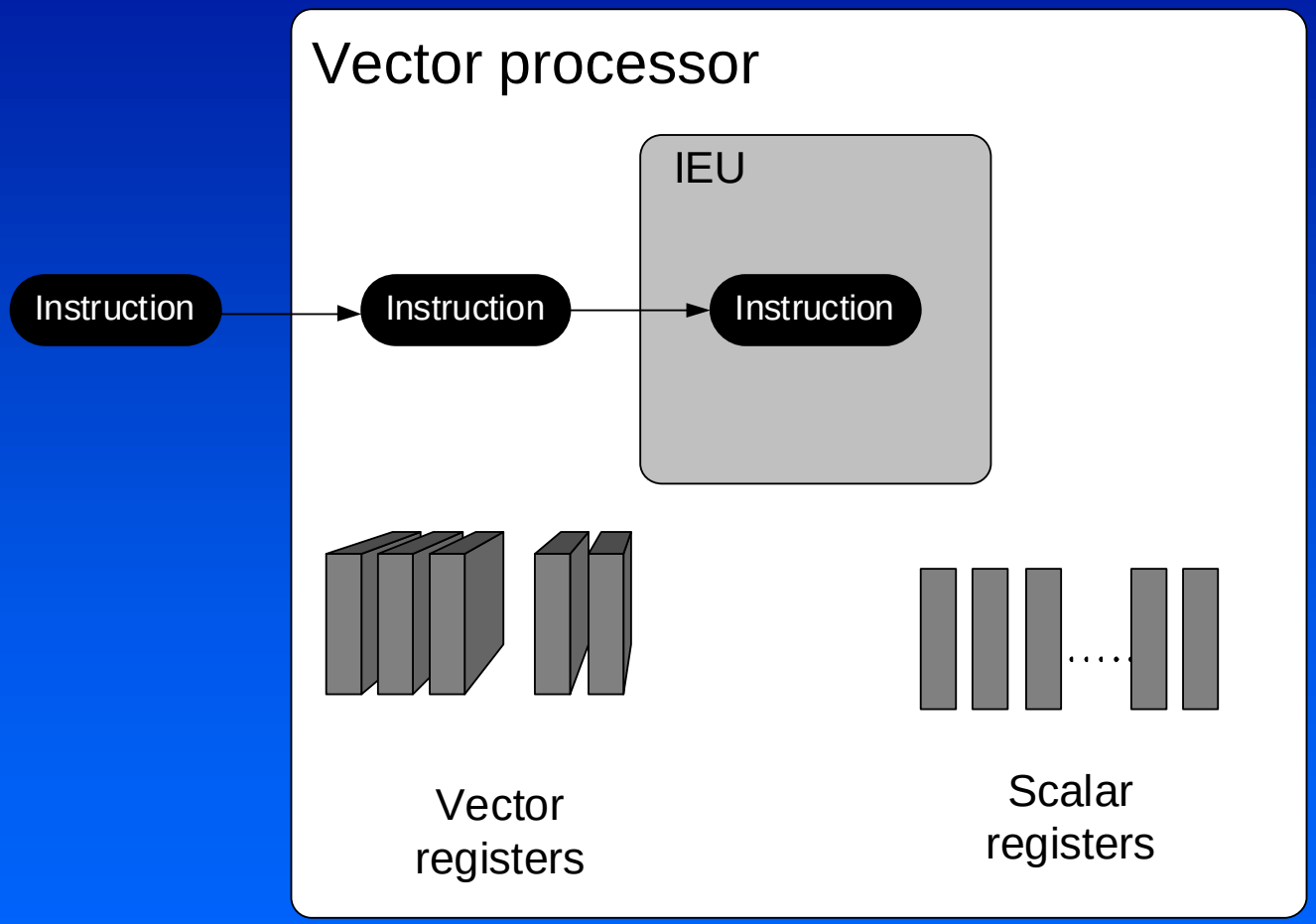
Serial scalar processor



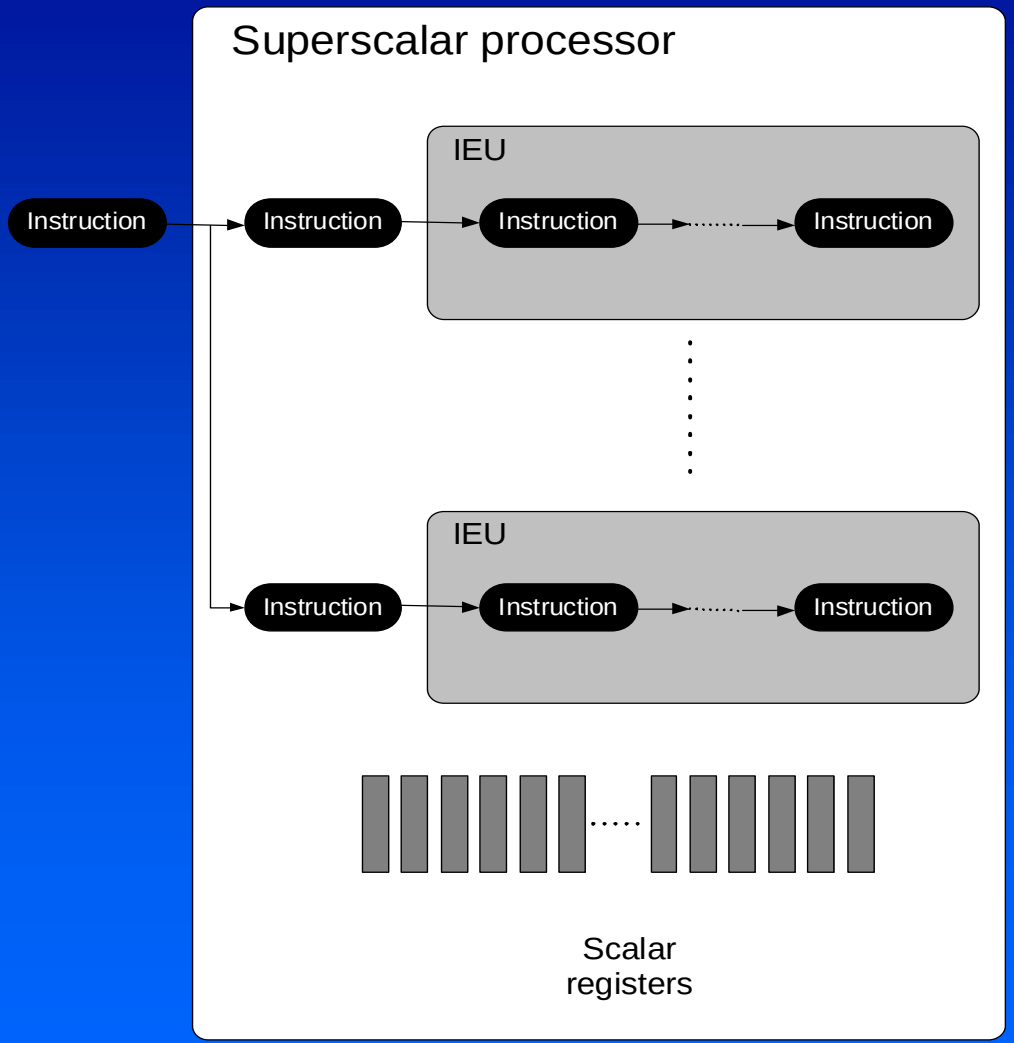
Main programming tools:

- C, Fortran 77

Vector processor



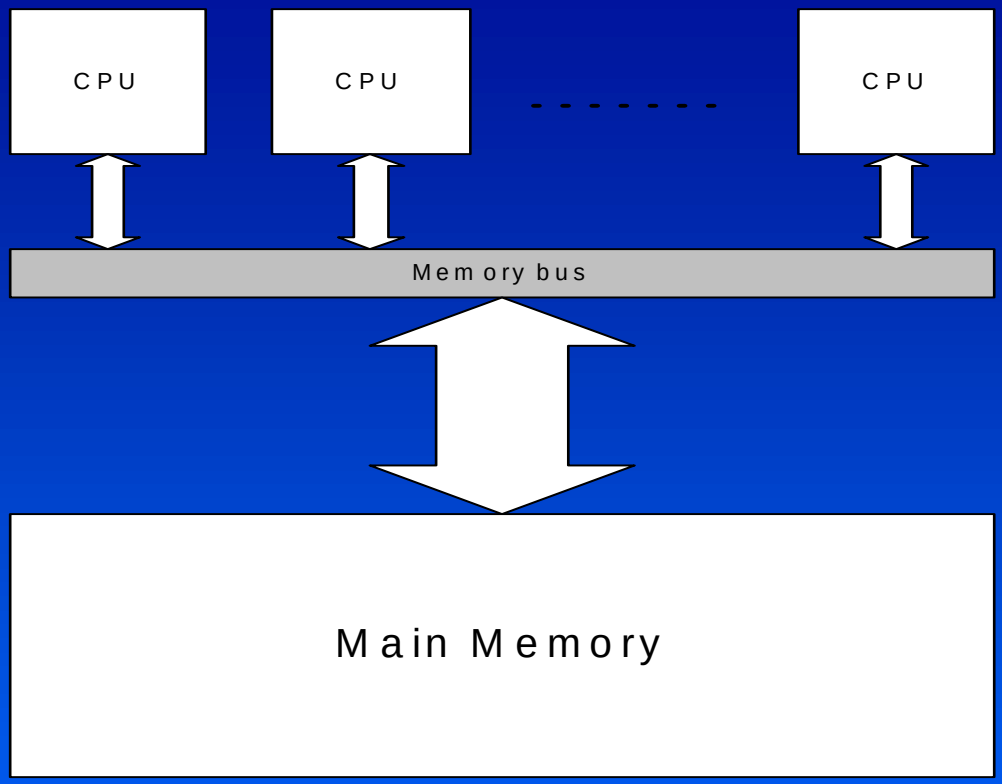
Superscalar processor



Programming tools for vector and superscalar processors

- Optimizing C and Fortran 77 compilers
- Array computation libraries (BLAS)
- Parallel languages (Fortran 90, C[])

Shared-memory multiprocessor



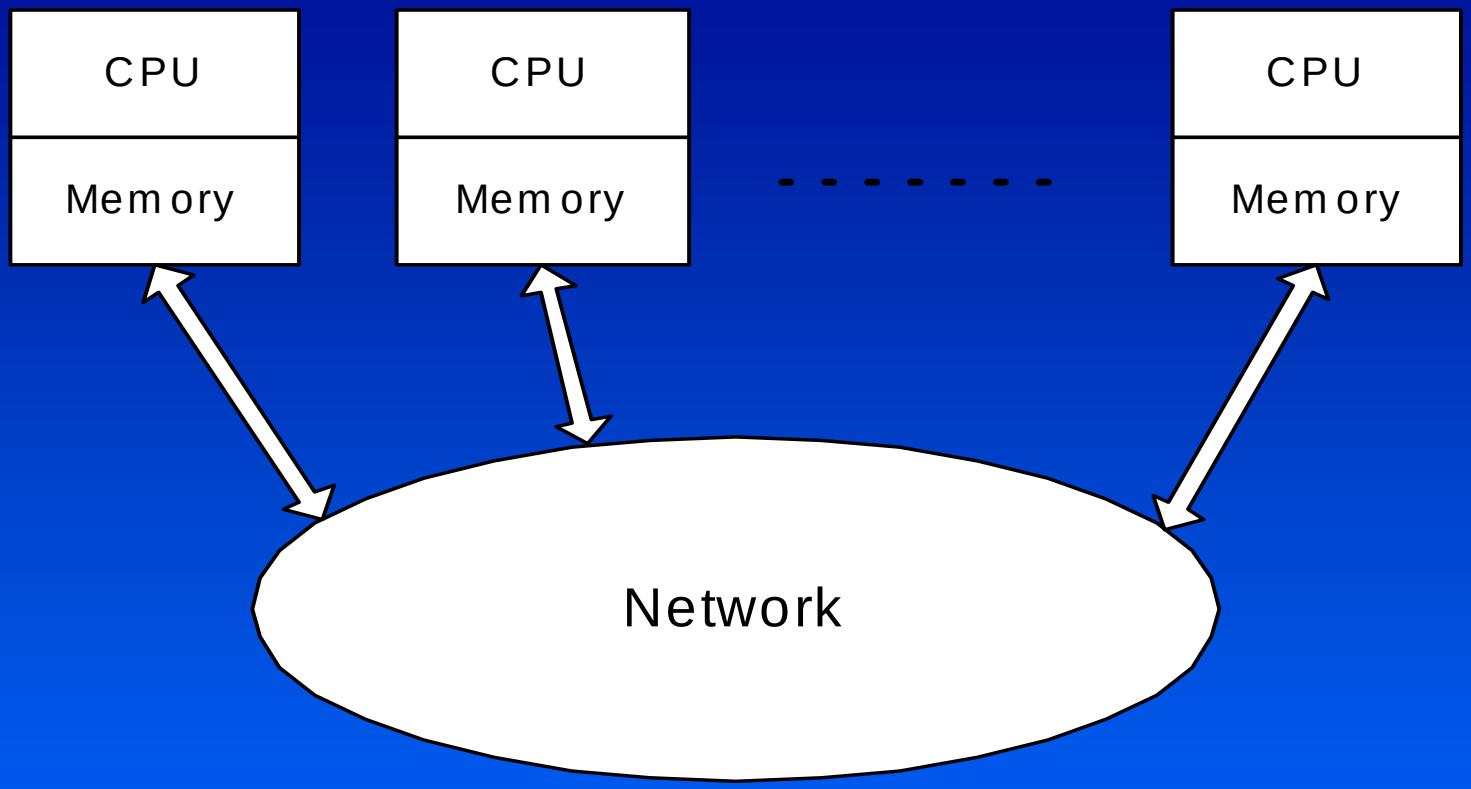
Parallel programming models

- Parallel threads of control sharing memory with other threads
- Parallel processes not sharing memory and interacting via message passing.

Programming tools for SMPs

- Optimizing C and Fortran 77 compilers
- Thread libraries (POSIX threads, etc.)
- Parallel languages
  - Fortran 95
    - Fortran 90 superset
  - Open MP
    - Standard high-level extensions to the most popular serial languages implementing the thread model of parallel programming
- Tools for message-passing parallel programming

Distributed-memory multiprocessor



- Scalable parallel architecture
- Supercomputers and clusters
- Message-passing parallel programming

Programming tools for MPPs

- Message-passing libraries (MPI - Message-Passing Interface, PVM - Parallel Virtual Machine)
- High Performance Fortran (HPF)

MPI

- Standardized in 1995 as MPI 1.1
- Widely implemented including high quality free implementations (LAM MPI, MPICH, etc.)
- Supports modular parallel programming
- Bindings - C, Fortran 77, C++

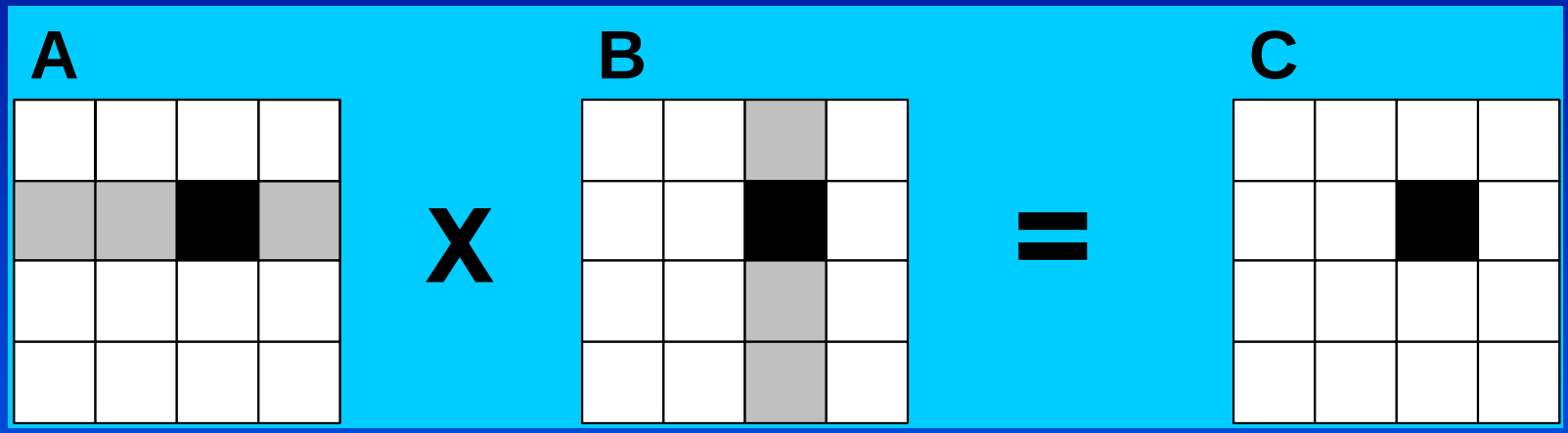
HPF

- Standardized as HPF 1.1 in 1994
- An extension of Fortran 90
  - With pseudo-directives specifying (block-cyclic) distribution of arrays over processors
- An easy parallel programming model
- A very difficult language to compile
  - Usually compiled into Fortran 77/90 + MPI
- Example. Let us consider an HPF application for parallel multiplication of two dense *nxn* matrices on *p* processors
  - The application implements 2D block algorithm

HPF (ctd)

```
      REAL, DIMENSION(1000,1000):: A, B, C
!HPF$ PROCESSORS p(4,4)
!HPF$ DISTRIBUTE (BLOCK,BLOCK) ONTO p:: A, B, C
!HPF$ INDEPENDENT
      DO J=1,1000
!HPF$   INDEPENDENT
        DO I=1,1000
          A(I,J)=1.0
          B(I,J)=2.0
        END DO
      END DO
!HPF$ INDEPENDENT
      DO J=1,1000
!HPF$   INDEPENDENT
        DO I=1,1000
          C(I,J)=0.0
          DO K=1,1000
            C(I,J)=C(I,J)+A(I,K)*B(K,J)
          END DO
        END DO
      END DO
      END PROGRAM
```

HPF (ctd)



- To compute its *C* slice, each processor requires the corresponding row of squares of the *A* matrix and column of squares of the *B* matrix
  - Receives from each of $\sqrt{p}-1$ horizontal and $\sqrt{p}-1$ vertical neighbours n^2/p matrix elements

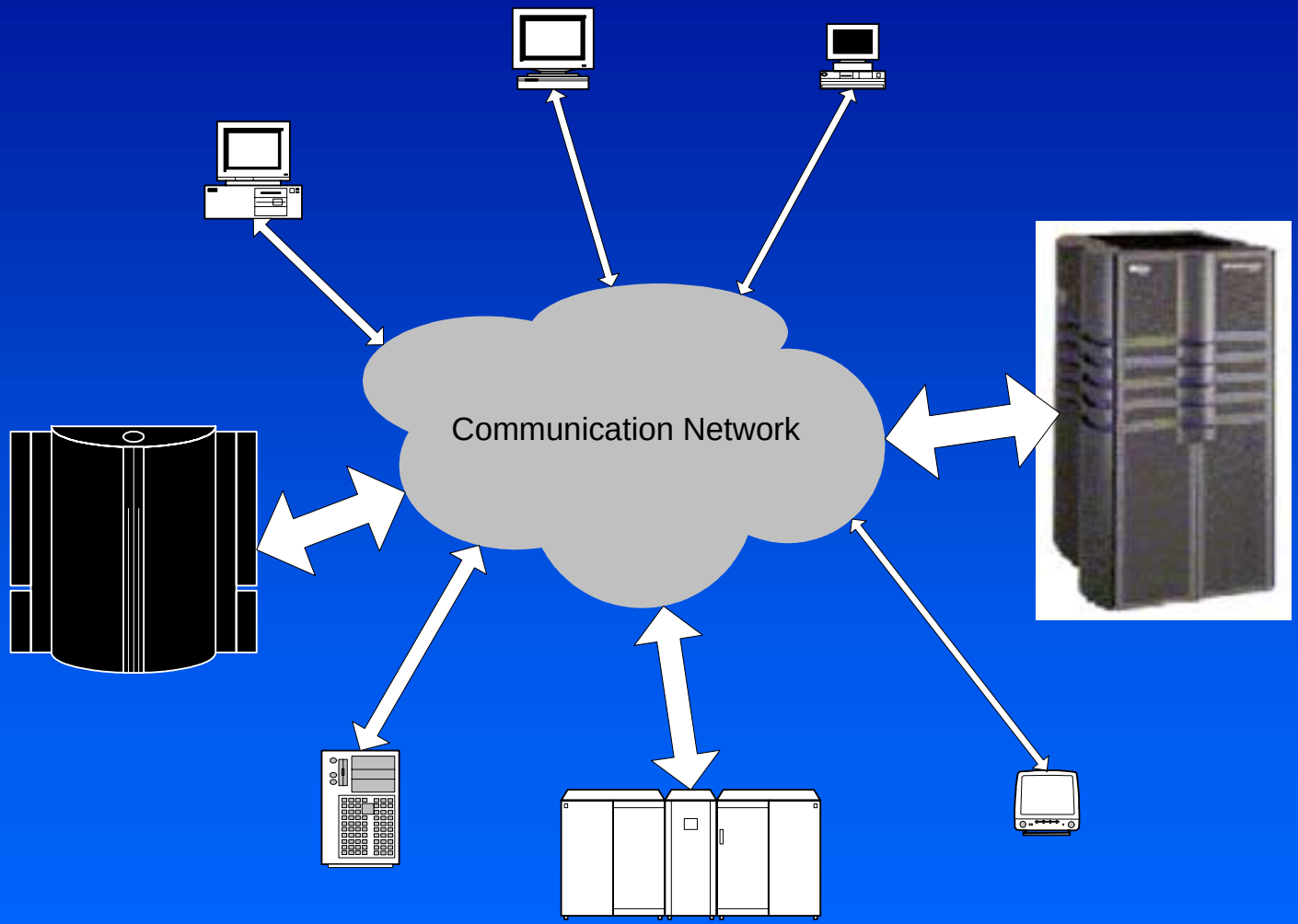
HPF (ctd)

- A clever HPF compiler will generate message passing code implementing the communication pattern depicted in the picture
- The minimization of inter-processor communication
  - The main optimisation performed by an HPF compiler
  - Not a trivial problem
  - No HPF constructs/directives helping the compiler to solve the problem
    - Therefore, HPF is considered a difficult language to compile

HPF (ctd)

- Many real HPF compilers
  - Will generate a message-passing program, where each process sends its blocks of *A* and *B* to *all* other processes
    - » This guarantees that each process receives all the elements of *A* and *B*, it needs to compute its elements of *C*
    - » This universal scheme involves a good deal of redundant communications
      - Sending and receiving data never used in computation

Network of computers



Programming Model

- NoCs seem very similar to MPPs at a first glance
  - Provides a number of processors
    - not sharing global main memory, and
    - interconnected via a communication network
- The most natural programming model
  - Parallel *processes*
    - each running on a separate processor
    - using message passing to communicate with the others

Network of Computers (ctd)

- Why are NoCs practically not used for parallel computing?
  - Parallel programming for NoCs is much more difficult than parallel programming for MPPs
    - NoCs are not designed and manufactured for HPC
    - A typical NoC is a naturally developed computer system
      - General purpose
      - Developed incrementally, for a relatively long time
      - Irregularity, heterogeneity, and instability are their inherent features

Network of Computers (ctd)

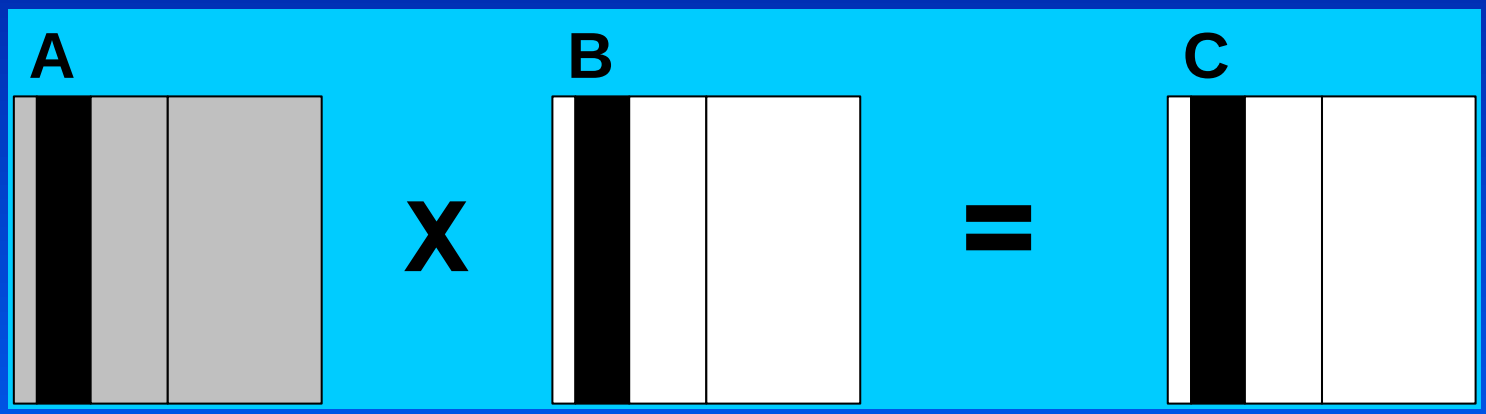
- Three main sources of the difficulties
  - The heterogeneity of processors
  - The communication network itself
    - not designed for high performance parallel computing
  - The multi-user nature of NoCs
    - not a strongly centralized computer system
    - consists of relatively autonomous computers
    - each computer is used and administered independently
    - different components are not as strongly integrated and controlled as in MPPs

Processors Heterogeneity

- Processors of different architectures
  - The processors run at different speeds
    - A good parallel program for MPPs evenly distributes workload
    - Ported to *heterogeneous cluster*, the program will align the performance with the slowest processor
    - A good parallel application for a NoC must distribute computations unevenly
      - taking into account the difference in processor speed
      - Ideally, the volume of computation performed by a processor should be proportional to its speed

Processors Heterogeneity (ctd)

- Example. Multiplication of two dense *nxn* matrices *A*, *B* on a *p*-processor heterogeneous cluster.



- ✓ Matrices A, B, and C unevenly partitioned in one dimension
- ✓ The area of the slice mapped to each processor is proportional to its speed
- ✓ The slices mapped onto a single processor are shaded black
- ✓ During execution, this processor requires all of matrix A (shaded gray)

Processors Heterogeneity (ctd)

- This algorithm cannot be implemented in HPF 1.1
  - No way to specify a heterogeneous distribution of arrays
- HPF 2.0 addresses the problem
  - `GEN_BLOCK` extends the `BLOCK` distribution with the ability to explicitly specify the size of each individual block
- The following HPF program
  - Multiplies two dense square 1000x1000 matrices on a 4-processor cluster, whose processors have relative speeds 2, 3, 5, and 10
  - The programmer must explicitly specify the exact distribution of the arrays across processors

Processors Heterogeneity (ctd)

```
      PROGRAM HETEROGENEOUS
      INTEGER, DIMENSION(4), PARAMETER::
     &M = (/ 100, 150, 250, 500 /)
      REAL, DIMENSION(1000,1000):: A, B, C
!HPF$ PROCESSORS p(4)
!HPF$ DISTRIBUTE (*, GEN_BLOCK(M)) ONTO p:: A,B,C
!HPF$ INDEPENDENT
      DO J=1,1000
!HPF$   INDEPENDENT
        DO I=1,1000
          A(I,J)=1.0
          B(I,J)=2.0
        END DO
      END DO
!HPF$ INDEPENDENT
      DO J=1,1000
!HPF$   INDEPENDENT
        DO I=1,1000
          C(I,J)=0.0
          DO K=1,1000
            C(I,J)=C(I,J)+A(I,K)*B(K,J)
          END DO
        END DO
      END DO
END
```

Processors Heterogeneity (ctd)

- The `REDISTRIBUTE` directive
  - Introduced in HPF 2.0
  - Distributes the arrays, based on a mapping array whose values are computed at runtime
  - Either the programmer or a user of the application must explicitly specify the data distribution
- The efficiency of the application strongly depends on the accuracy of estimation of the relative speed of processors of the heterogeneous cluster
  - If this estimation is not accurate, the load of processors will be unbalanced

Processors Heterogeneity (ctd)

- Two processors of the same architecture only differing in the clock rate
  - No problem to accurately estimate their relative speed
    - The relative speed will be the same for any application
- Processors of different architectures
  - Differ in everything
    - set of instructions, number of IEUs, number of registers, memory hierarchy, size of each memory level, and so on
  - May demonstrate different relative speeds for different applications

Processors Heterogeneity (ctd)

- Example. Consider two implementations of a 500x500 matrix Cholesky factorisation:
  - The code

```
for(k=0; k<500; k++) {
  for(i=k, lkk=sqrt(a[k][k]); i<500; i++)
    a[i][k] /= lkk;
  for(j=k+1; j<500; j++)
    for(i=j; i<500; i++)
      a[i][j] -= a[i][k]*a[j][k];
}
```

    estimated the relative speed of SPARCstation-5 and SPARCstation-20 as 10:9

Processors Heterogeneity (ctd)

- Example. (ctd)
  - The code

```
for(k=0; k<500; k++) {
  for(i=k, lkk=sqrt(a[k][k]); i<500; i++)
    a[i][k] /= lkk;
  for(i=k+1; i<500; i++)
    for(j=i; j<500; j++)
      a[i][j] -= a[k][j]*a[k][i];
}
```

    estimated their relative speed as 10:14
  - Routine `dptof2` from LAPACK, solving the same problem, estimated their relative speed as 10:10

Processors Heterogeneity (ctd)

- Heterogeneity of machine arithmetic
  - Different processors do not guarantee the same storage representation and the same results for operations on floating point numbers
    - the stopping criterion used by the most accurate processor may never be satisfied if it depends on data computed less accurately by other processors
  - The communication layer does not guarantee the exact transmittal of the floating-point value
    - overflow/underflow exceptions may occur during conversions, resulting in a failure of the communication

Ad Hoc Communication Network

- Typical communication network
  - Primary factors determining the structure
    - structure of the organisation, tasks that are solved, security requirements, construction restrictions, budget limitations, qualification of technical personnel, etc.
  - High performance computing
    - secondary factor if considered at all
  - Constantly developing
    - occasionally and incrementally
    - the communication network reflects the evolution of the organization rather than its current snapshot

Ad Hoc Communication Network (ctd)

- As a result, the common communication network
  - Far away from the ideal MPP communication network
    - Heterogeneous
    - May have links of low speed and/or narrow bandwidth
- Optimal distribution of computations and communications across a NoC
  - Much more difficult than across a *heterogeneous cluster*
    - larger size of the problem, $O(n^2)$
    - due to links of low speed/bandwidth, optimal distribution may be not across the entire NoC, making the problem complexity exponential

Multi-user decentralised system

- Unstable performance characteristics
  - Computers executing your parallel program
    - May be used for other computations
    - May be involved in other communications
  - Real performance of processors and communication links can dynamically change
  - A good parallel program for a NoC must be sensitive to dynamic variations of its workload

Multi-user decentralised system (ctd)

- High probability of resource failures
  - Fault tolerance is not a primary problem for MPPs
    - Small probability of resource failures
  - The probability of resource failures is much higher for NoCs
    - Programming systems supporting fault tolerance are needed
      - MPI-FT (transparent for programmers)
      - FT-MPI (extends MPI)

Summary of Programming Challenges

- Main features of an ideal parallel program for a NoC
  - Distributes computations and communications unevenly across processors and communications links
    - takes into account their actual performance demonstrated during the execution of the code of the program
    - the distribution is not static
      - may be different not only for different NoCs but even for different executions of the program on the same NoC
    - the program may find profitable to involve in computations not all available computers
    - In other words, the program must be *efficiently portable*

Summary of Challenges (ctd)

- The program keeps running even if some resources in the executing network fail
- The program takes into account differences in machine arithmetic on different computers
  - avoids erroneous behaviour caused by the differences

- Are MPI and HPF suitable for parallel programming networks of computers?

Summary of Challenges (ctd)

- HPF
  - Very basic support for parallel programming NoCs
    - means to specify uneven distribution of data across abstract HPF processors
      - programmers must calculate the best distribution
  - No means to control the mapping of abstract HPF processors to computers of the NoC
  - Does not address fault tolerance

Summary of Challenges (ctd)

- MPI
  - Low level programming tool => can be used for efficiently portable parallel programming NoCs
    - No specific support for programming NoCs
      - All specific code that make the application efficiently portable must be manually written by programmers
  - Standard MPI specification does not address fault tolerance
    - There are fault tolerant implementations of the standard MPI specification (MPI-FT)
    - There is an extension of standard MPI specification supporting explicit programming fault-tolerant MPI programs (FT-MPI)
  - Typically, multiprotocol communications are not supported

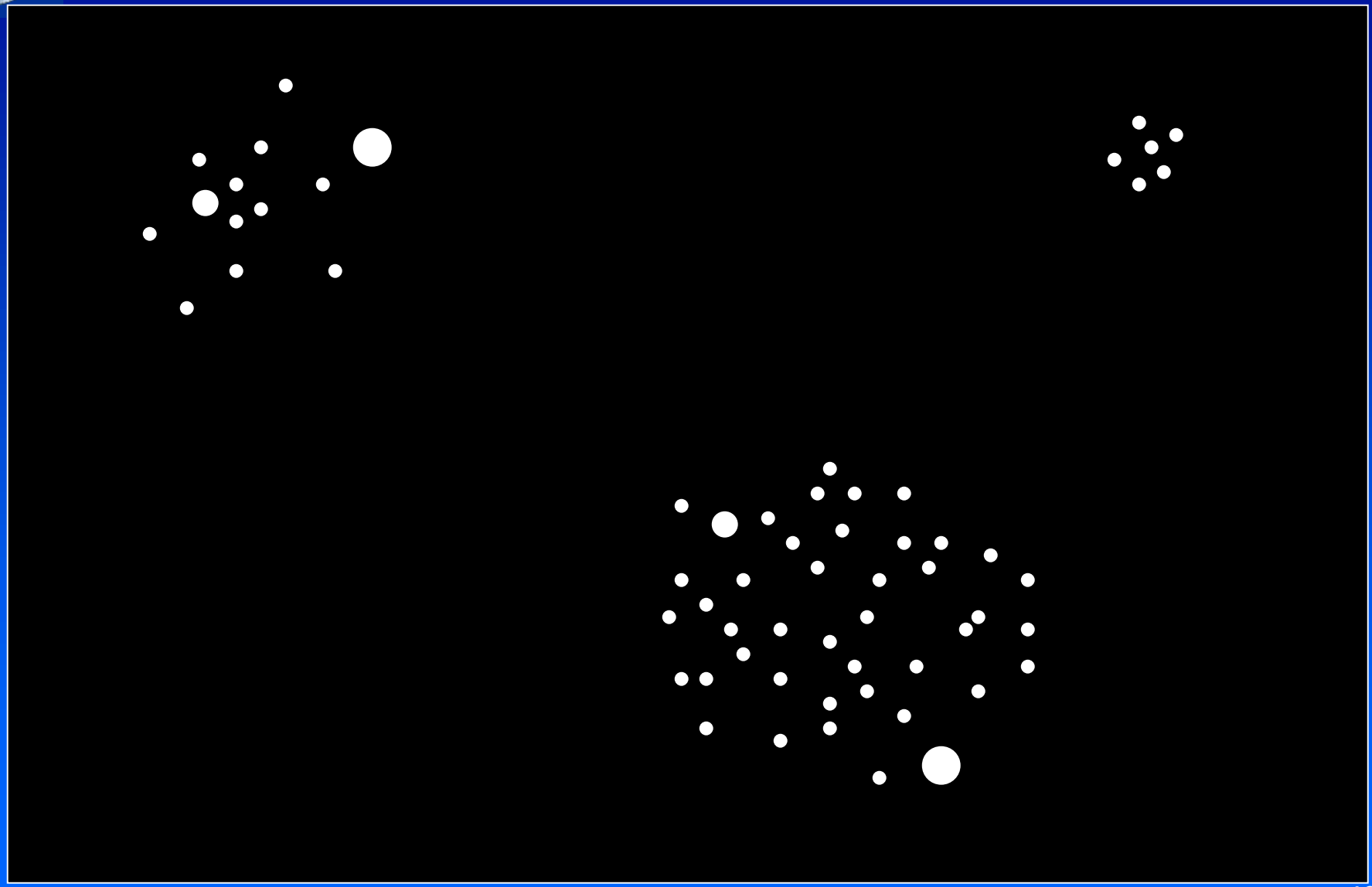
mpC

- mpC - a dedicated parallel programming language (an ANSI C superset) for high-performance computing on heterogeneous networks
- mpC allows writing portable parallel programs that dynamically distribute computations and communications over any particular executing network to provide the best efficiency

mpC (ctd)

- mpC program defines main features of the parallel algorithm, which influence its efficiency
  - Total number of participating processes
  - Total volume of computations to be performed by each of the processes
  - Total volume of data to be transferred between each pair of the processes
  - Interaction scenario during execution of the algorithm
- mpC programming system
  - Uses this information at run time to map processes of the parallel program to the executing network in such a way that ensures its best execution performance

Parallel N-body Simulation



Parallel N-body Simulation (ctd)

```
Initialization of galaxy on host-process
Scattering groups of bodies over processes
Parallel computing masses of groups
Interchanging the masses among processes
while(1) {
   Visualization of galaxy by host-process
   Parallel computing centers of gravity
   Interchanging the centers among processes
   Parallel updating groups
   Gathering groups on host-process
}
```

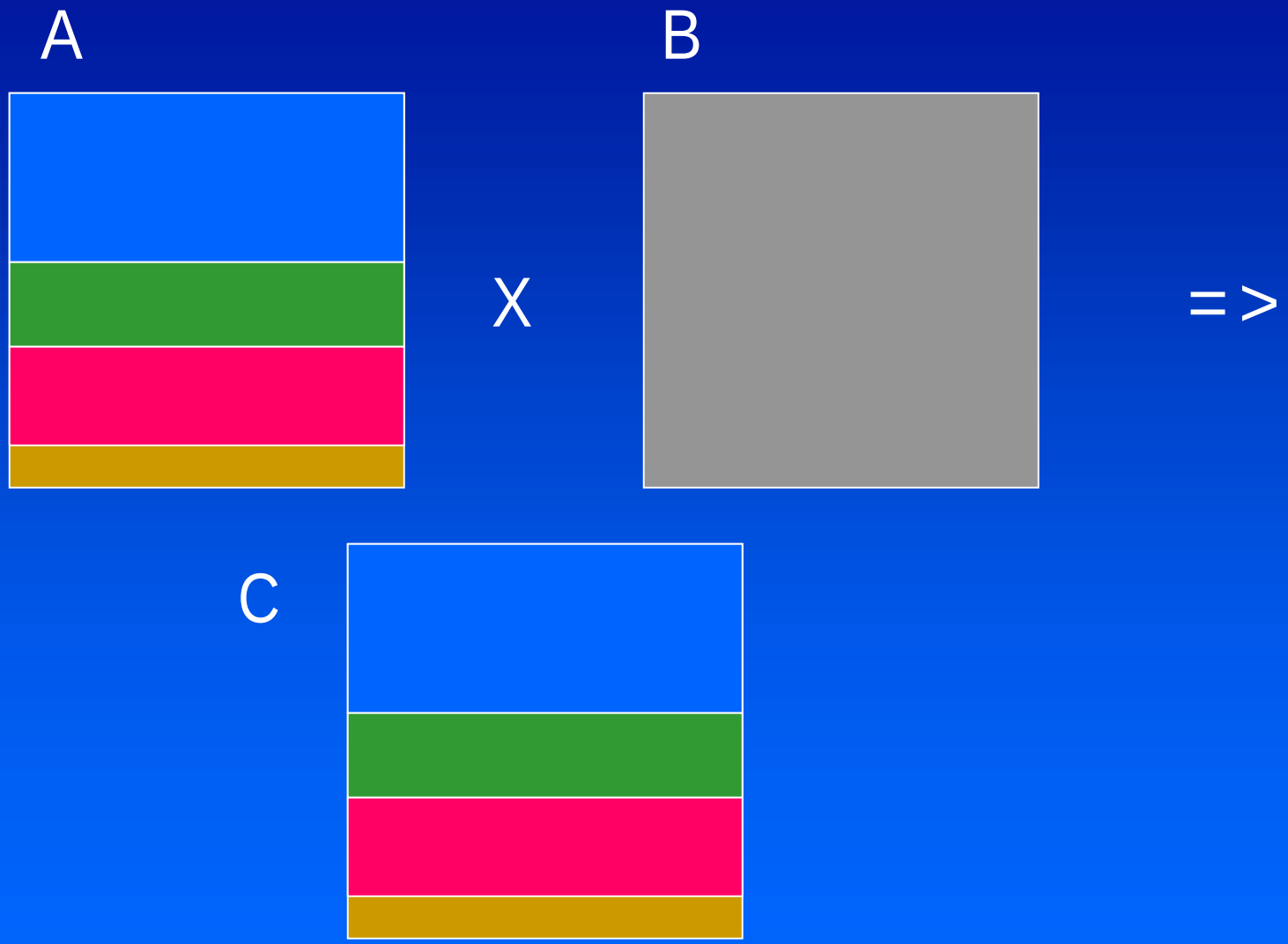
Parallel N-body Simulation (ctd)

```
nettype GalaxyNet (m, k, n[m]) {
   coord I=m;
   node {  I>=0:  bench * ( (n[I]/k) * (n[I]/k) );  };
   link   {  I>0  :   length * ( n[I] * sizeof(Body) )  [I]->[0];  };
   parent [0];
   scheme {
      int i;
      par (i=0; i<m; i++)  100%%  [i];
      par (i=1; i<m; i++)  100%%  [i]->[0];
   };
};
```

Parallel N-body Simulation (ctd)

```
...
void [*] main ( int [host]argc,  char ** [host]argv )
{
   ...
   TestGroup[] = (*Galaxy[0])[];
   recon   Update_group ( TestGroup, TestGroupSize ) ;
  {
      net  GalaxyNet (NofG, TestGroupSize, NofB)  g;
      ...
   }
}
```

Parallel Matrix Multiplication



Parallel Matrix Multiplication (ctd)

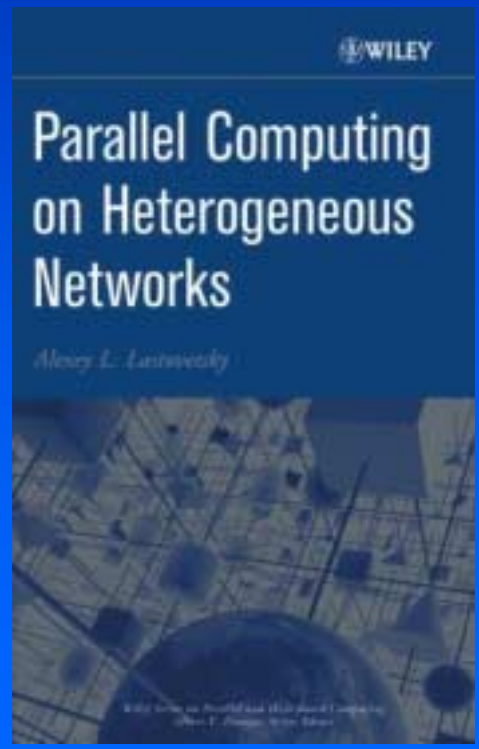
```
nettype AxB(int m, int k, int t, int n[m]) {
  coord I=m;
  node { I>=0: bench * ( n[I] / t ); };
  link (J=m) {
    J>0: length * ( ( k + n[J] ) * k * sizeof(float) )  [0] ->[J];
    I>0: length * ( n[I] * k * sizeof(float) )           [I]->[0];
  };
  parent [0];
  scheme {
    int i;
    for ( i=1;  i<m;  i++ )   100%%  [0]->[i];
    par ( i=0;  i<m;  i++ )  100%%  [i];
    par ( i=1;  i<m;  i++ )  100%%  [i]->[0];
  };
};
```

Parallel Matrix Multiplication (ctd)

```
...
recon SeqMult ( tA, tB, tC, M, N );
...
[host]: { ...
            for (j = 1; j <= nprocs; j++) {
              Partition(j, powers, nrows, N);
              t = timeof (net Star(j, N, M, nrows) w);
              if (t < min.t) { min.nprocs = j; min.t = t; }
            }
            nprocs = min.nprocs;
          }
...
Partition (nprocs, powers, nrows, N);
{
   net AxB(nprocs, N, M, nrows) w;
   ...
}
```

References

Alexey Lastovetsky. Parallel Computing on Heterogeneous Networks. John Wiley & Sons, 423 pp., June 2003.



Conclusion

- Parallel computing is shifting to network computing
- Parallel programming tools can power existing software suites for both local and global computer networks