GPU Programming and Productivity in Software Development

MANUEL ARENAZ

UNIVERSIDADE DA CORUÑA
Associate Professor
manuel.arenaz@udc.es

Appentra
CTO & Co-founder
manuel.arenaz@appentra.com

V Jornadas de Supercomputación y Avances en Tecnología COMPUTAEX

Cáceres, November 19-20, 2012

http://www.cenits.es/noticias/241012-fundacion-computaex-organiza-v-jornadas-supercomputacion-avances-tecnologia

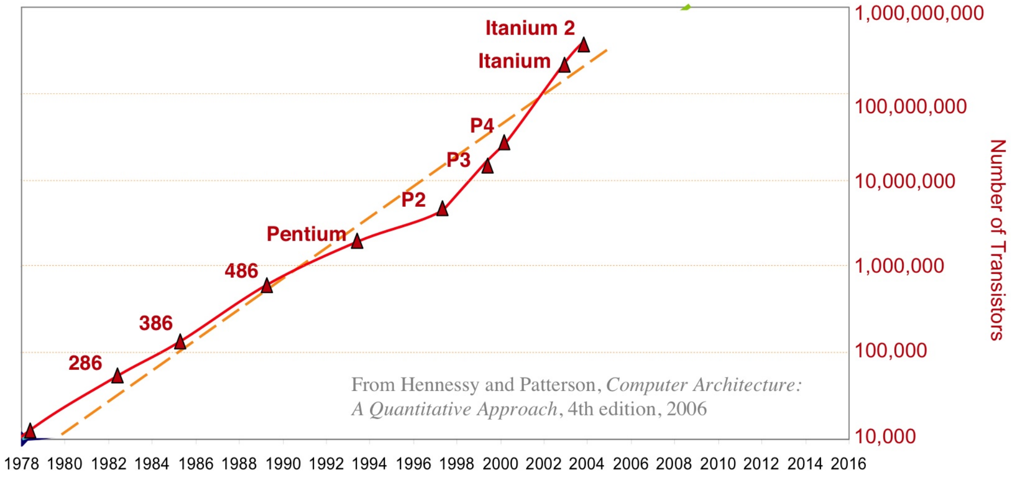
Agenda

- □ Do I really need parallelism?
- □ The HPC software marketplace can be organized from the productivity viewpoint?
- □ What do I need to learn to write HPC Apps?
- □ How can I be more productive?
- □ Are there “Design Patterns” to help writing HPC Apps?
- □ Can you give an example of a parallel design pattern?
- □ Are there Hw-independent key concepts?
- □ Are there frequently used design patterns for HPC Apps?
- □ Can you propose a development methodology for GPU programming?
- □ Can you meassure productivity?
- □ Conclusions

Do I really need parallelism?

Historical Perspective

- □ Moore Law:
  The number of transistors in a chip doubles every 18 months



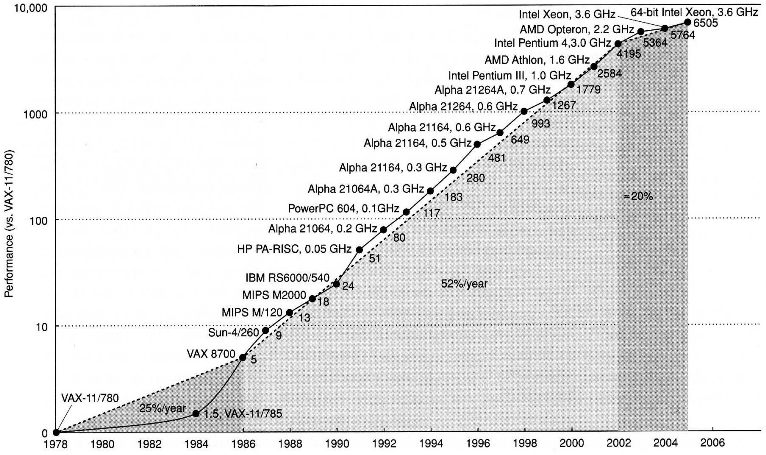
From Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, 4th edition, 2006

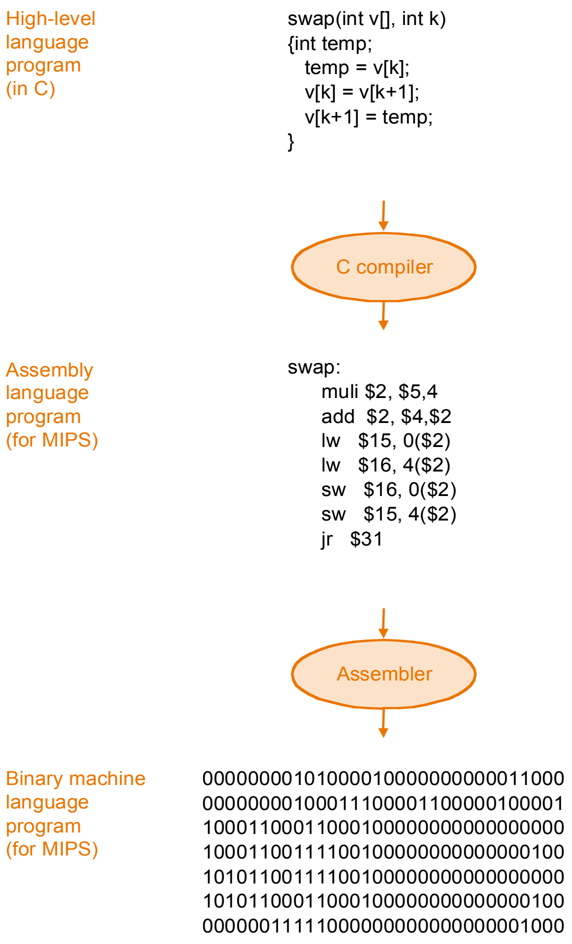
How do we use transisors **productively**

How do we program modern systems more **productively**

1st Software Crisis (60s-70s)

- Problems:
  - Programming in assembly language
  - Computers can execute programs much more complex
- Solution:
  - Higher level of abstraction: Procedural programming
  - Portability & Performance
  - New languages & tools: C, Fortran





High-level language program (in C)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

Assembly language program (for MIPS)

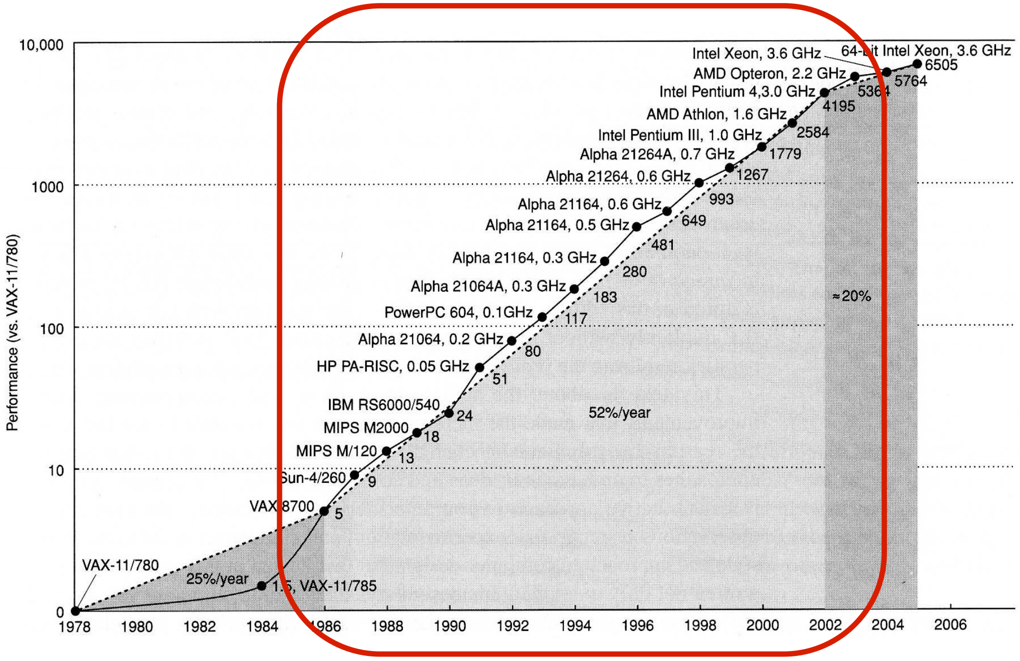
```
swap:
      muli $2, $5,4
      add  $2, $4,$2
      lw   $15, 0($2)
      lw   $16, 4($2)
      sw   $16, 0($2)
      sw   $15, 4($2)
      jr   $31
```

Binary machine language program (for MIPS)

```
00000000101000010000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Performance for Free (80s-90s)

- □ Advances in computer architecture ensure that performance doubles every 2 years!



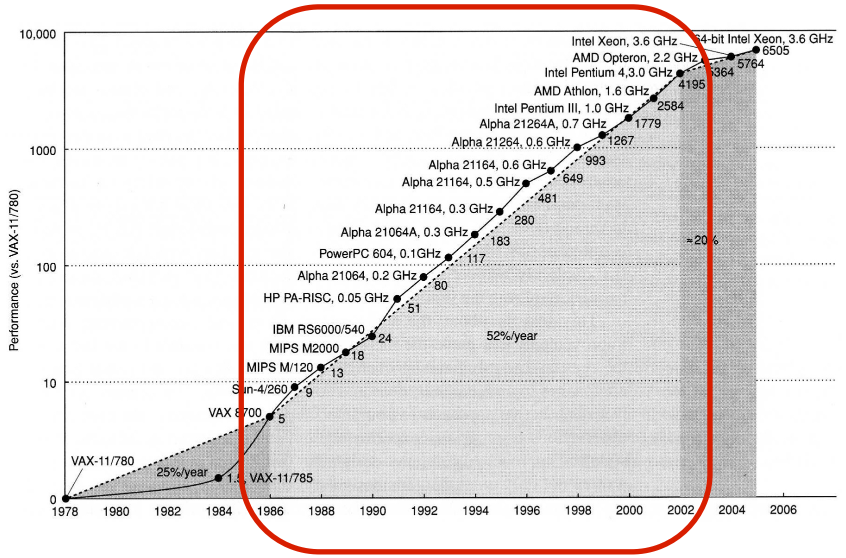
Source: “J.L. Hennessy y D.A. Patterson: *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 4ª Edition, 2008”

- □ Performance is a Hw issue

- □ Sw issues: Recompile your program and run faster

2nd Software Crisis (80s-90s)

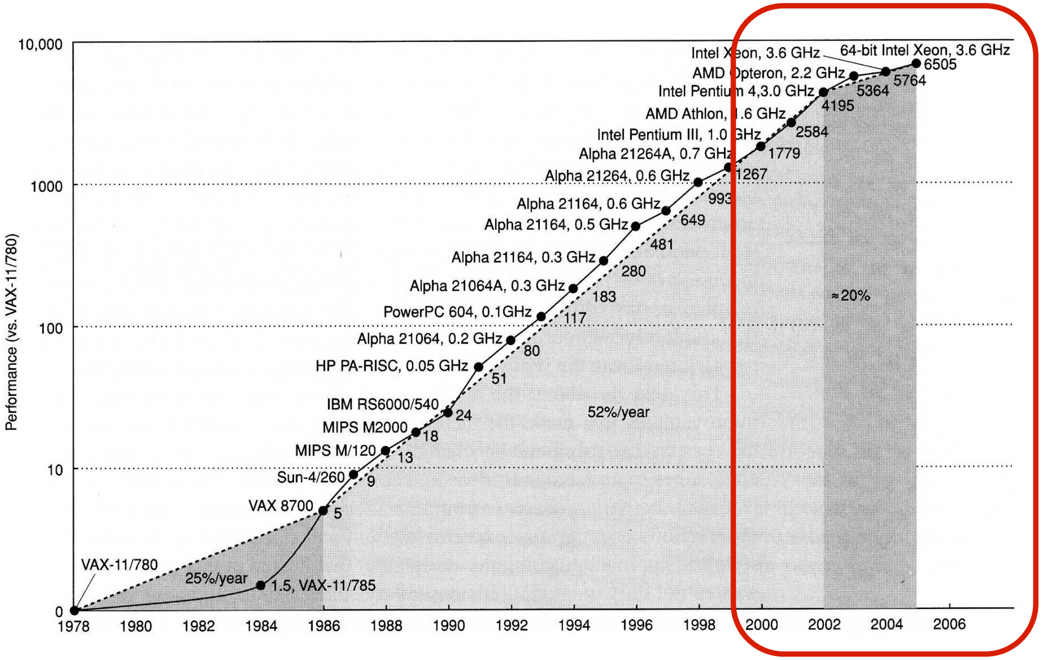
- Problems:
  - Impossible to build and maintain big, complex and robust Apps developed by hundreds of programmers
  - Again, computers can execute programs much more complex



- Solution:
  - Higher level of abstraction: Object-oriented programming
  - New languages & tools: C++, C#, Java
  - Portability… but don’t care about performance!

“Poor” Performance Gain (>2002)

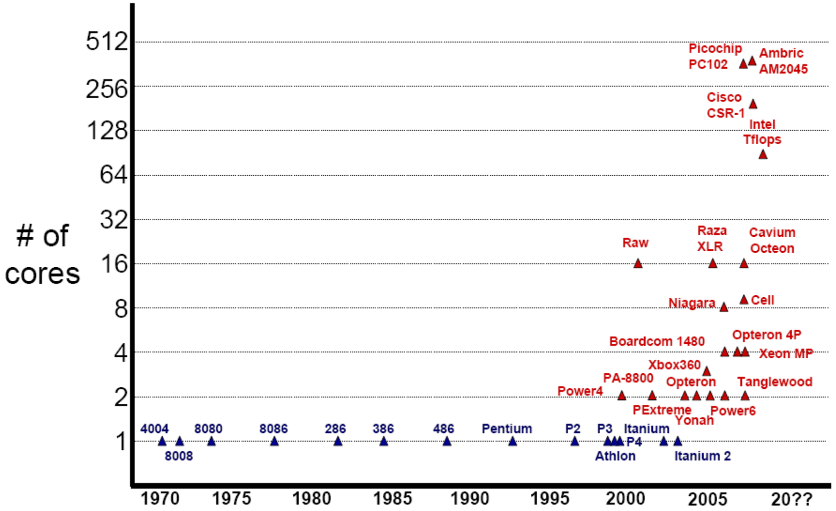
- Key challenges for the manufacturers:
  - Heat dissipation problems
  - Not enough instruction level parallelism (ILP) in applications
  - Increasing gap between CPU speed and memory speed



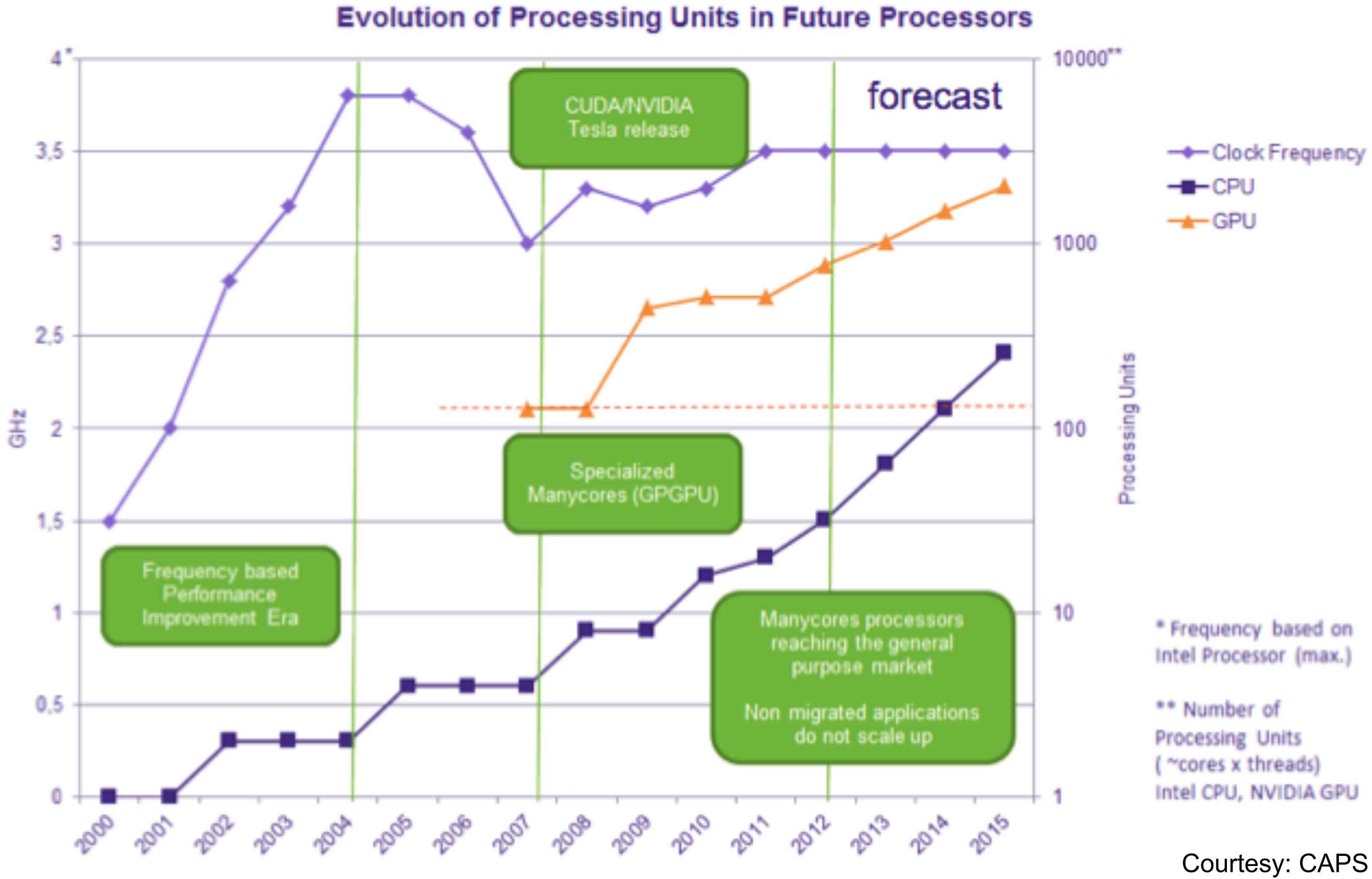
“Poor” Performance Gain (>2002)

- Manufacturers (Intel, AMD, IBM, ARM) change their strategy
  - Transistors are not used to design more complex processors
  - New “multi-core processors” are lunched to the market.
- New Moore Law:

  The number of cores in a processor will double every 18 months.



Hw Marketplace Forecast

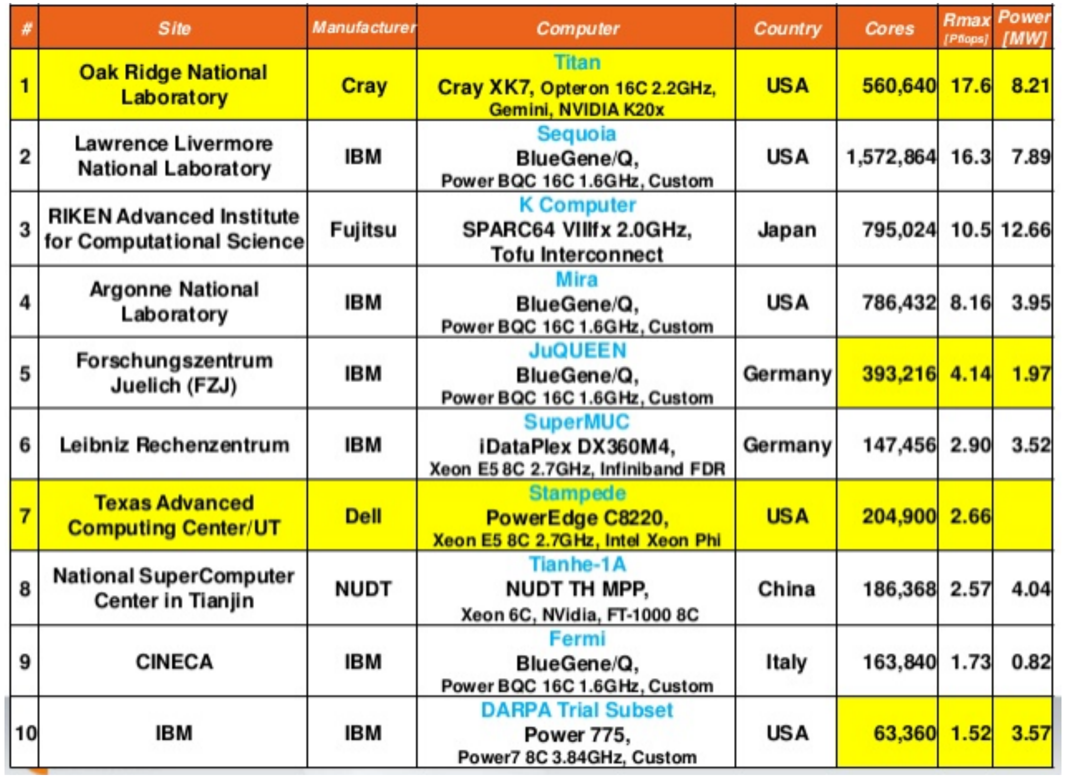


* Frequency based on Intel Processor (max.)

** Number of Processing Units (~cores x threads) Intel CPU, NVIDIA GPU

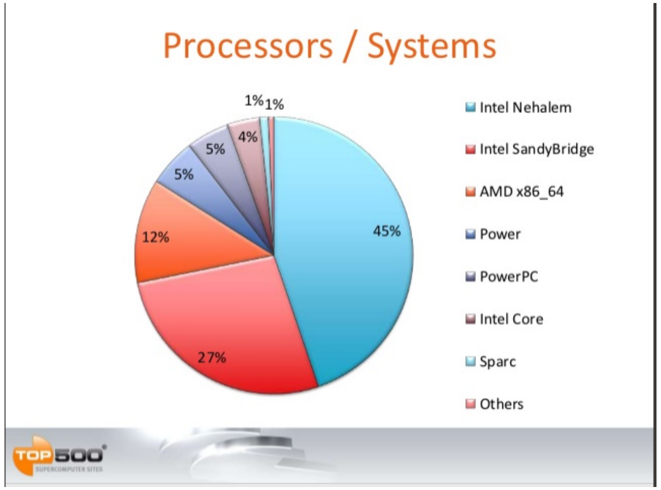
Courtesy: CAPS HMPP CoC Europe "Write once, deploy many-core"

TOP500 40th November 2012

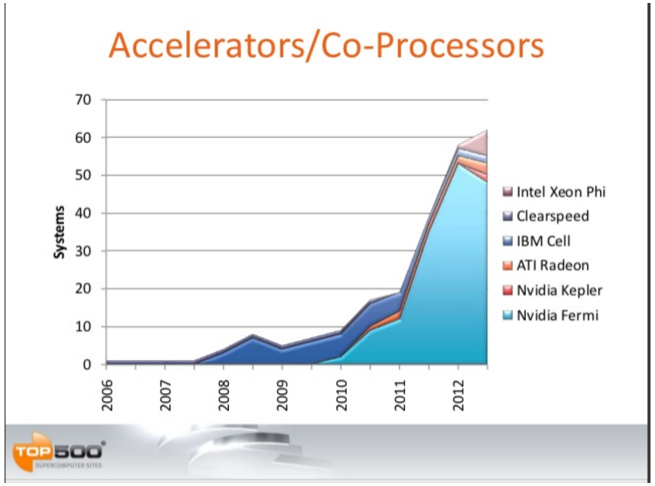


#	Site	Manufacturer	Computer	Country	Cores	Rmax [Pflops]	Power [MW]
1	Oak Ridge National Laboratory	Cray	Titan Cray XK7, Opteron 16C 2.2GHz, Gemini, NVIDIA K20x	USA	560,640	17.6	8.21
2	Lawrence Livermore National Laboratory	IBM	Sequoia BlueGene/Q, Power BQC 16C 1.6GHz, Custom	USA	1,572,864	16.3	7.89
3	RIKEN Advanced Institute for Computational Science	Fujitsu	K Computer SPARC64 VIIIfx 2.0GHz, Tofu Interconnect	Japan	795,024	10.5	12.66
4	Argonne National Laboratory	IBM	Mira BlueGene/Q, Power BQC 16C 1.6GHz, Custom	USA	786,432	8.16	3.95
5	Forschungszentrum Juelich (FZJ)	IBM	JuQUEEN BlueGene/Q, Power BQC 16C 1.6GHz, Custom	Germany	393,216	4.14	1.97
6	Leibniz Rechenzentrum	IBM	SuperMUC iDataPlex DX360M4, Xeon E5 8C 2.7GHz, Infiniband FDR	Germany	147,456	2.90	3.52
7	Texas Advanced Computing Center/UT	Dell	Stampede PowerEdge C8220, Xeon E5 8C 2.7GHz, Intel Xeon Phi	USA	204,900	2.66	
8	National SuperComputer Center in Tianjin	NUDT	Tianhe-1A NUDT TH MPP, Xeon 6C, NVidia, FT-1000 8C	China	186,368	2.57	4.04
9	CINECA	IBM	Fermi BlueGene/Q, Power BQC 16C 1.6GHz, Custom	Italy	163,840	1.73	0.82
10	IBM	IBM	DARPA Trial Subset Power 775, Power7 8C 3.84GHz, Custom	USA	63,360	1.52	3.57

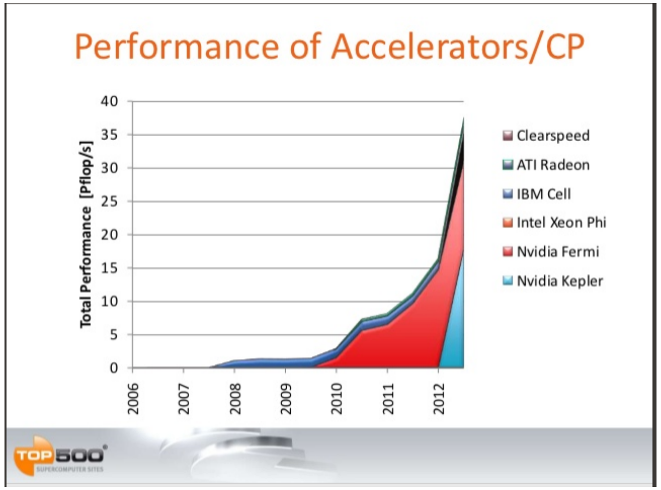
TOP500 40th November 2012



Processors / Systems

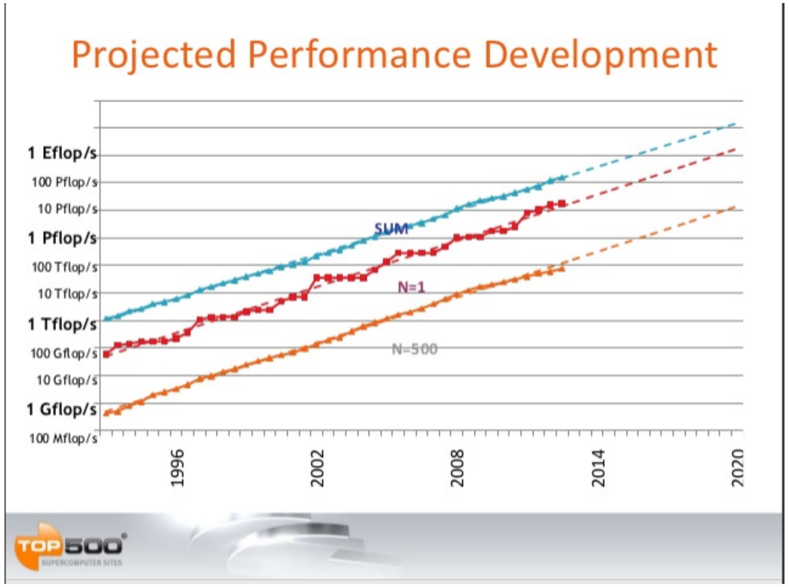


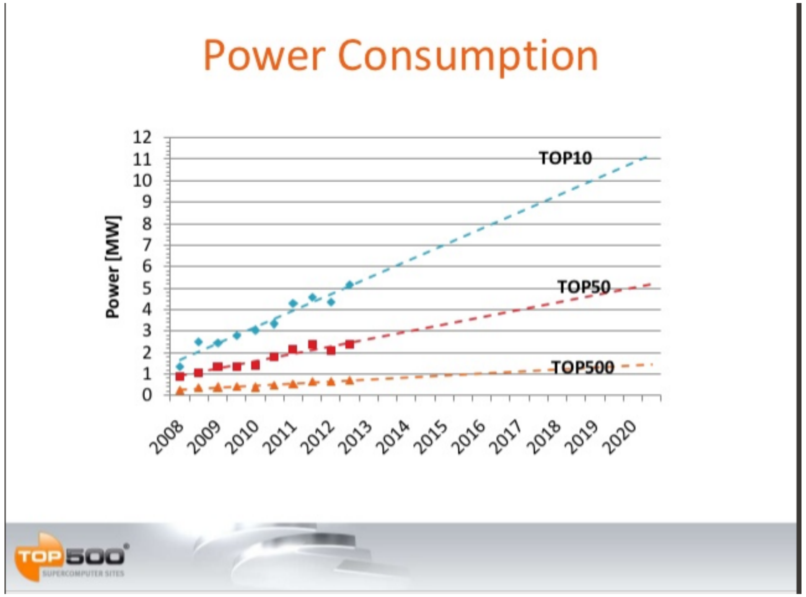
Accelerators/Co-Processors



Performance of Accelerators/CP

TOP500 40th November 2012





3rd Software Crisis

IDC
Analyze the Future
PRACE
ETP 4 HPC
PlanetHPC
THE FRAMEWORK PROGRAMME FOR RESEARCH AND INNOVATION
HORIZON 2020
HPC USER FORUM
SEVENTH FRAMEWORK PROGRAMME
ICT Information and Communication Technologies
INTERNATIONAL EXASCALE SOFTWARE PROJECT
ROADMAP1.1
18
EESI
HiPEAC
COMPILATION ARCHITECTURE

Urgent need for **new software**
to increase **productivity**

3rd Software Crisis

3rd Software Crisis

Applications **underutilize** all the computational power

3rd Software Crisis

- The abstraction level of current languages&tools provides great freedom to programmers…, but they are not aware of hardware features
  - 30-year-old programs run today with much better performance!

- No “for-free” performance gain any more
  - The performance of mono-core Apps will not improve as in the past!
  - The hardware features are now exposed to the programmer!

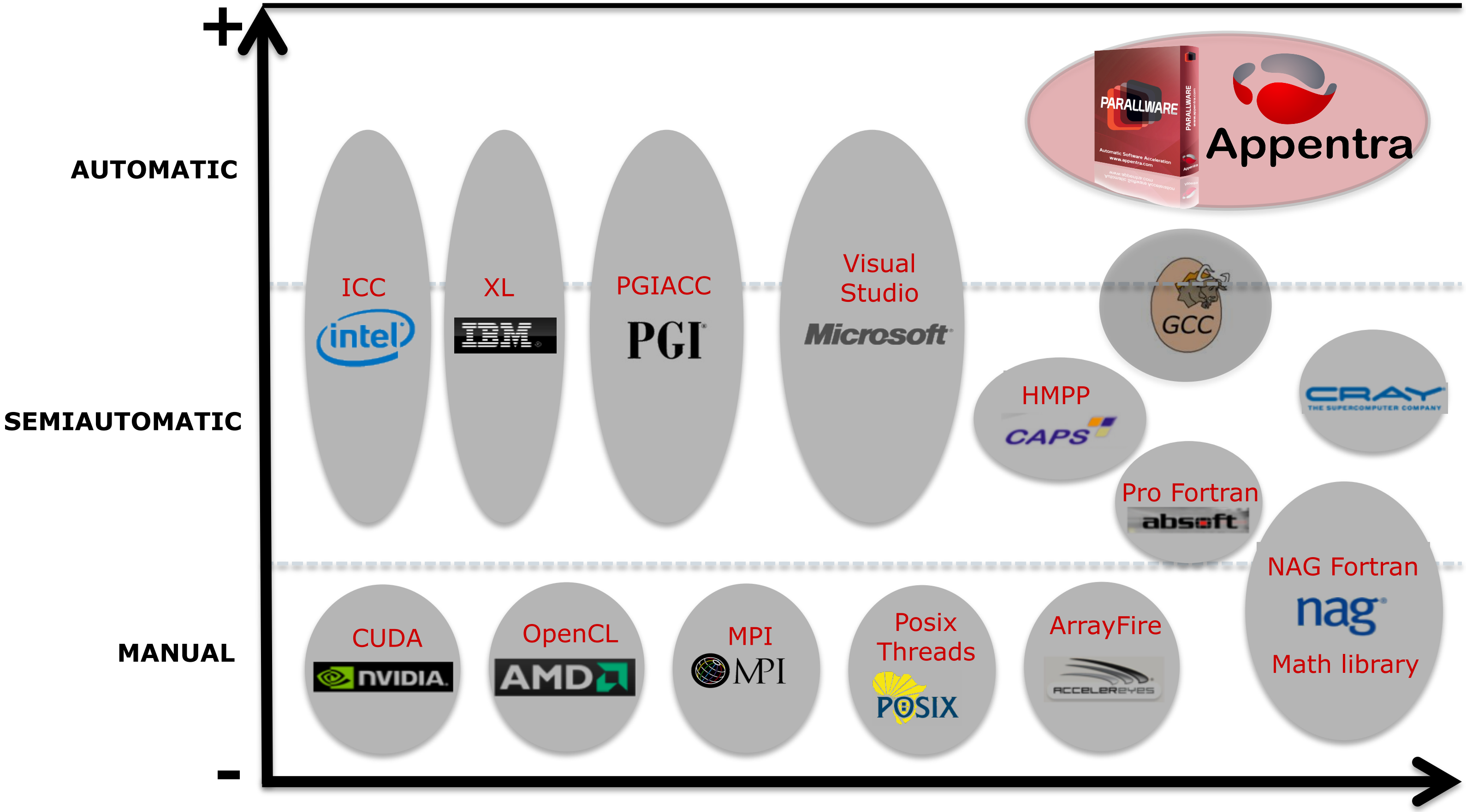
- New Apps still…
  - … demand more-and-more performance
  - … demand portability & maintainability

Parallel-Programming & HPC techniques are a **must** to exploit modern systems

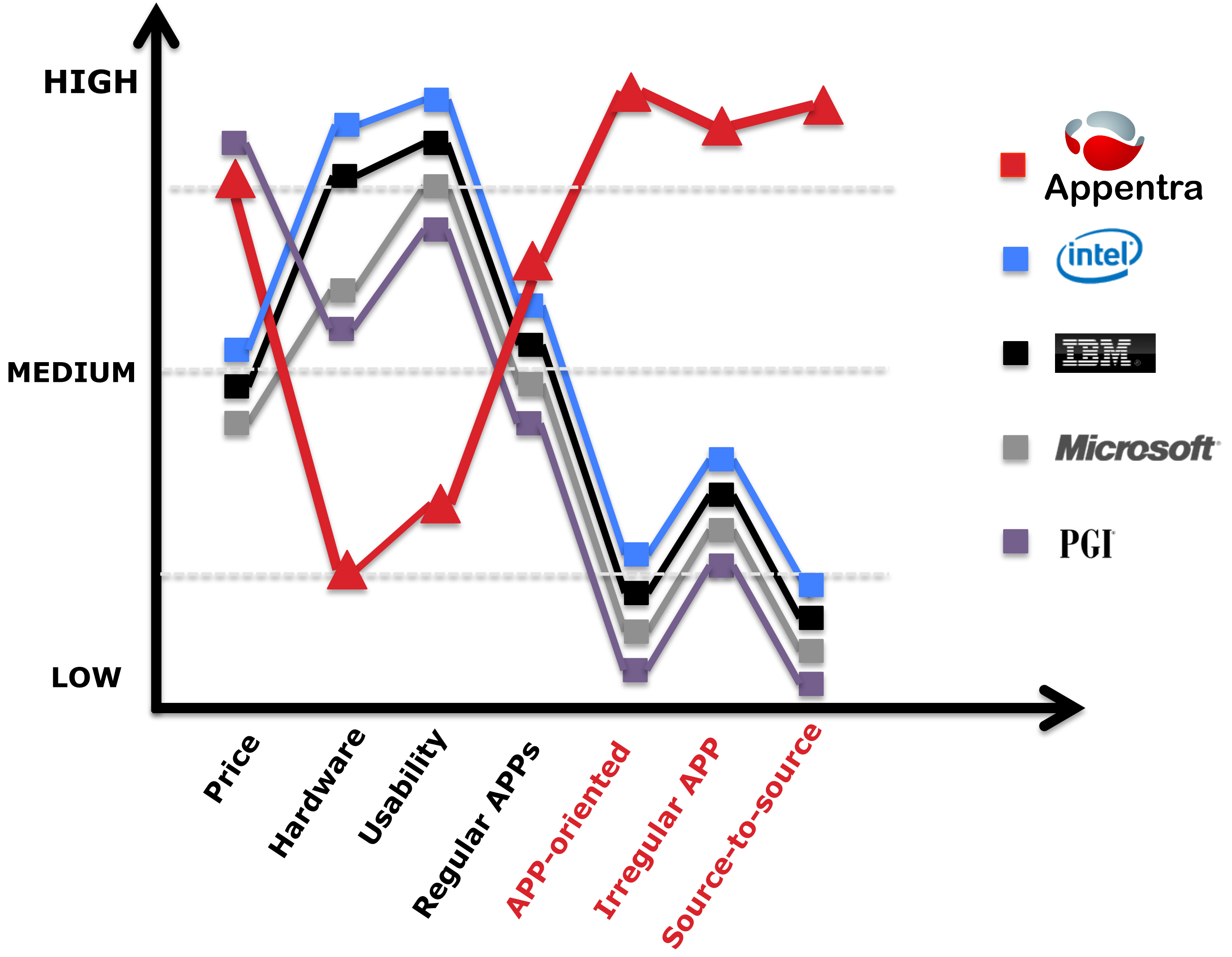
Do I really need parallelism?

The HPC software marketplace can be organized from the productivity viewpoint?

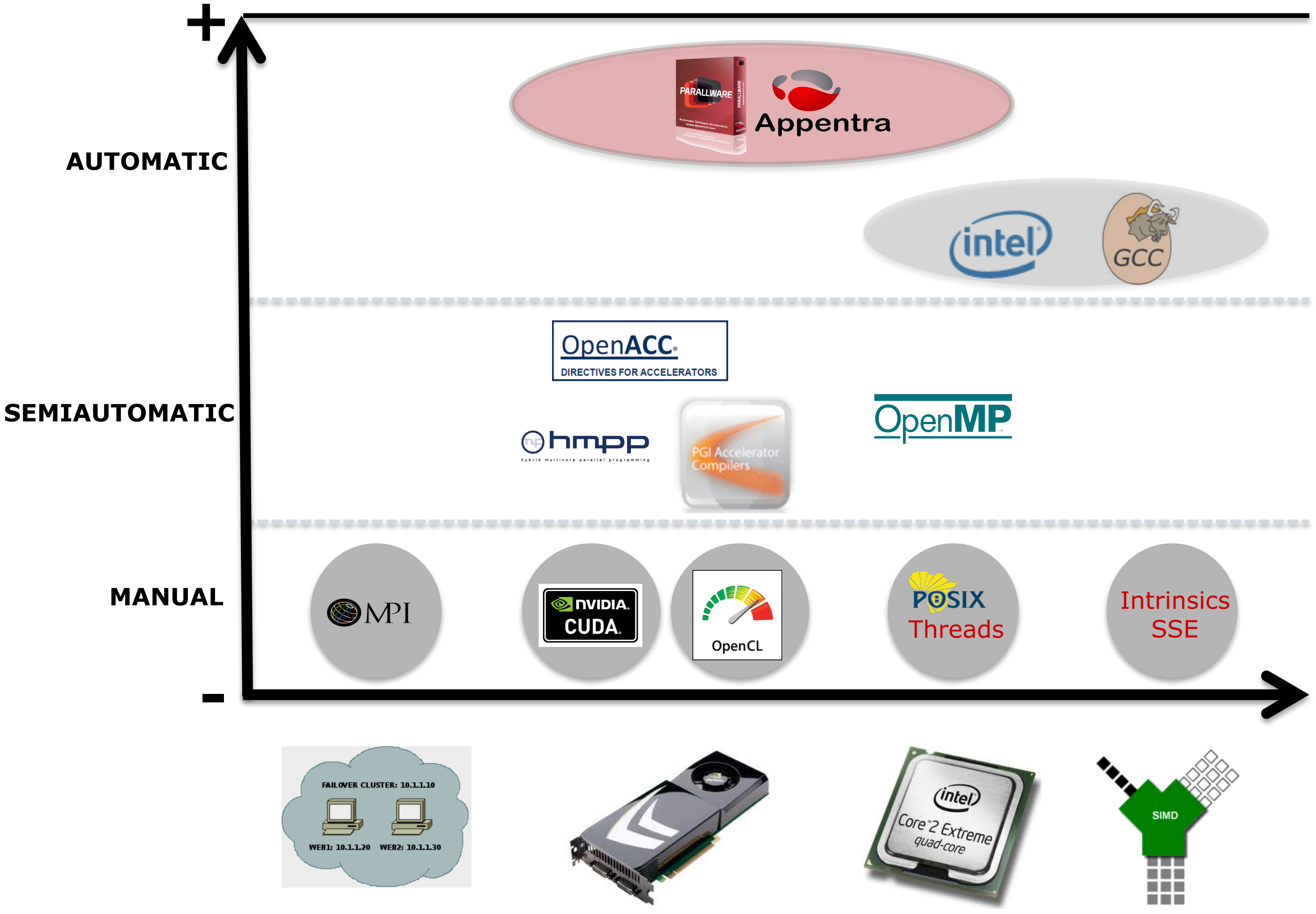
Software Marketplace



Technological Comparison



Software Technologies



+

AUTOMATIC

SEMIAUTOMATIC

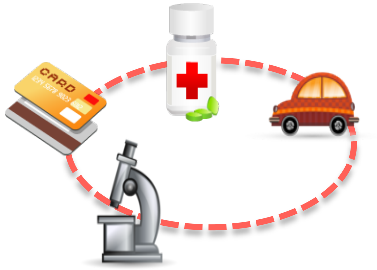
MANUAL

-

The HPC software marketplace can be organized from the productivity viewpoint?

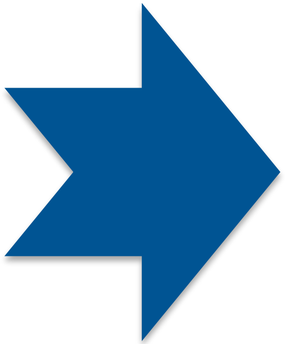
What do I need to learn to write HPC Apps?

Science & Engineering

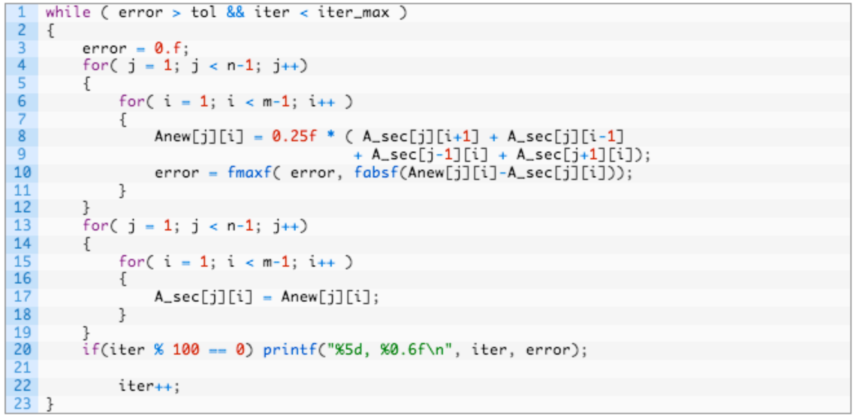


Industrial/Scientific Domain

- Knowledge
- Expertise
- Methods
- Techniques
- Tools



- Coding&Testing
- Validation



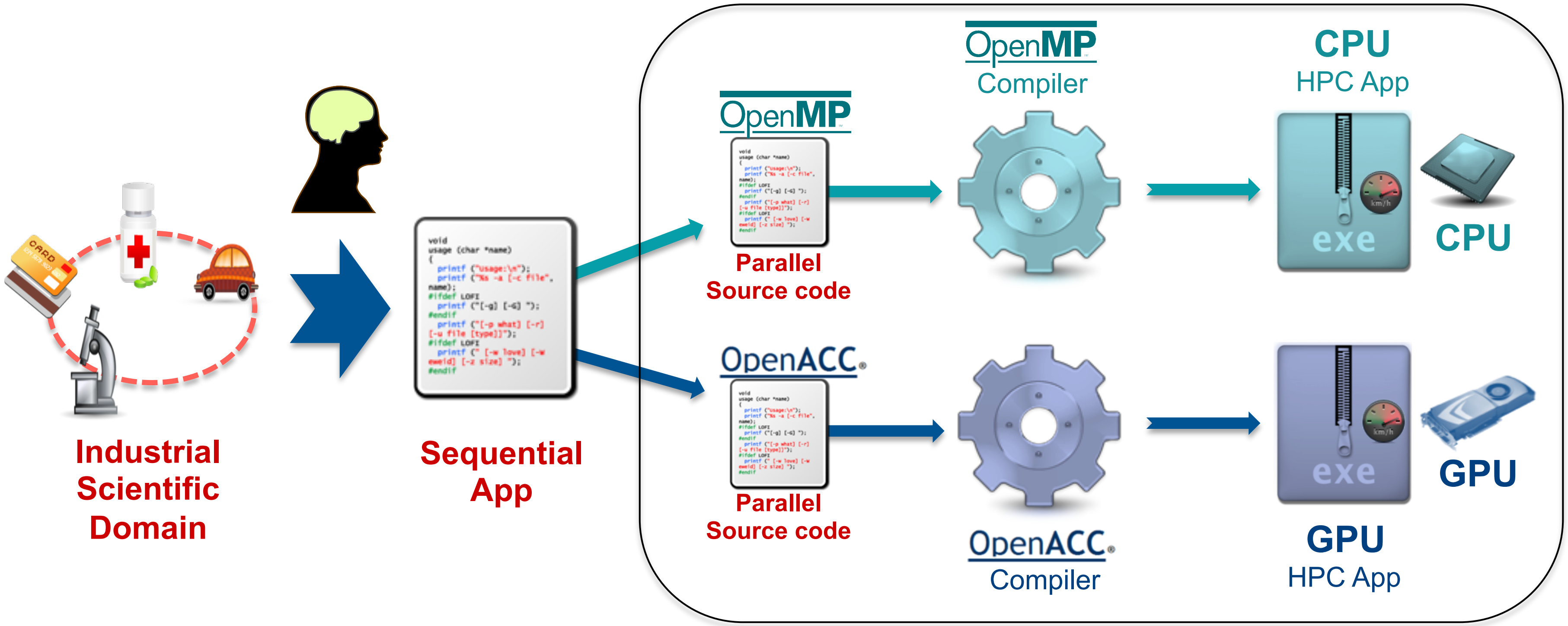
```
while ( error > tol && iter < iter_max )
{
    error = 0.f;
    for( j = 1; j < n-1; j++)
    {
        for( i = 1; i < m-1; i++ )
        {
            Anew[j][i] = 0.25f * ( A_sec[j][i+1] + A_sec[j][i-1]
                                 + A_sec[j-1][i] + A_sec[j+1][i]);
            error = fmaxf( error, fabsf(Anew[j][i]-A_sec[j][i]));
        }
    }
    for( j = 1; j < n-1; j++)
    {
        for( i = 1; i < m-1; i++ )
        {
            A_sec[j][i] = Anew[j][i];
        }
    }
    if(iter % 100 == 0) printf("%5d, %0.6f\n", iter, error);

        iter++;
}
```

Sequential App

- Knowledge in Computer Science
- Algorithm design and implementation
- Programming languages
- Compilers, libraries & other tools

Engineer
Scientist
Researcher

Science & Engineering… & HPC



```
void
usage (char *name)
{
  printf ("Usage:\n");
  printf ("%s -a [-c file",
name);
#ifdef LOFI
  printf ("[-g] [-G] ");
#endif
  printf ("[-p what] [-r]
[-u file [type]]");
#ifdef LOFI
  printf (" [-w love] [-w
eweid] [-z size] ");
#endif
```

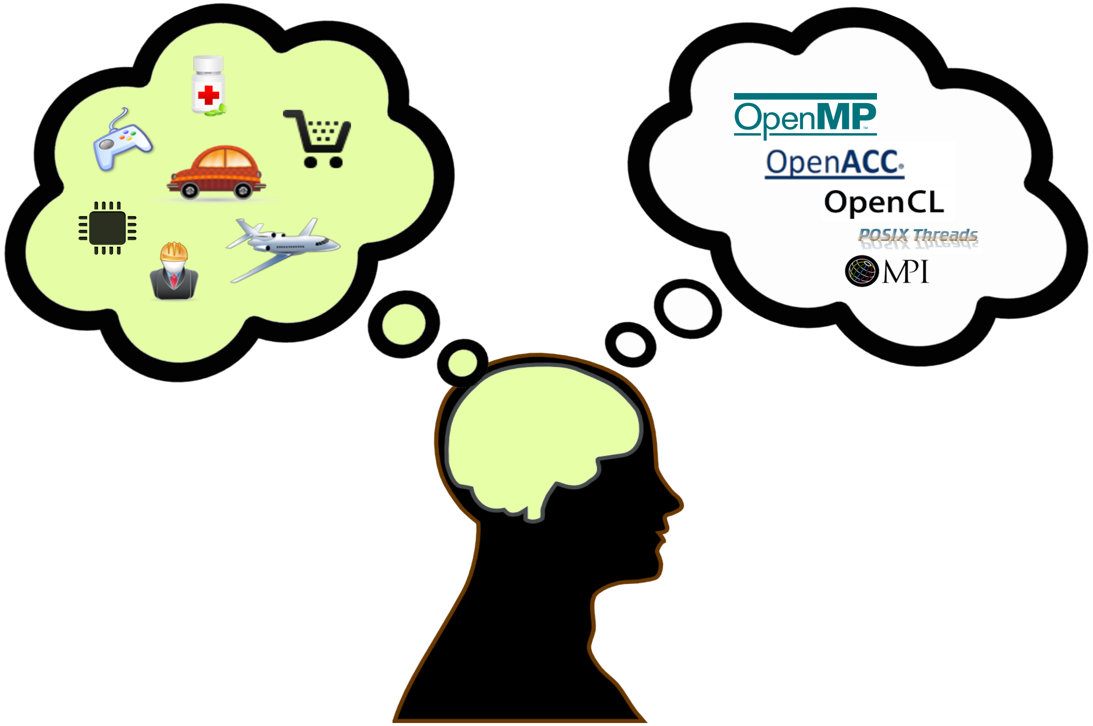
HPC App

- **Knowledge in Parallel Programming**
- **Knowledge in Computer Architecture**
- **Parallelization-enabling tools (profilers, libraries, compilers, languages…)**

Learn to be a Superman

□ You need deep knowledge of your business…

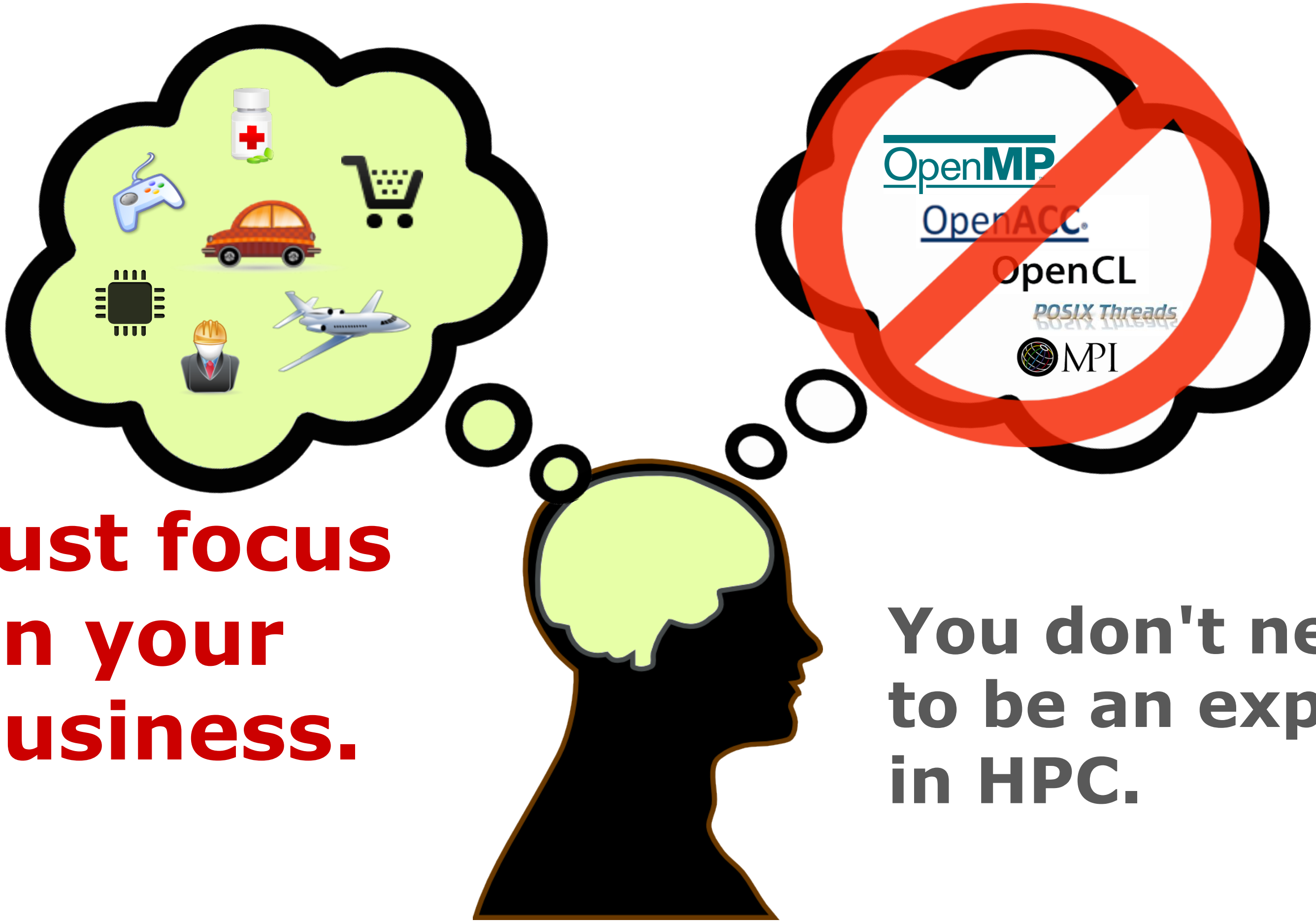
□ But also deep knowledge of Hw & HPC.



What do I need to learn to write HPC Apps?

How can I be more productive?

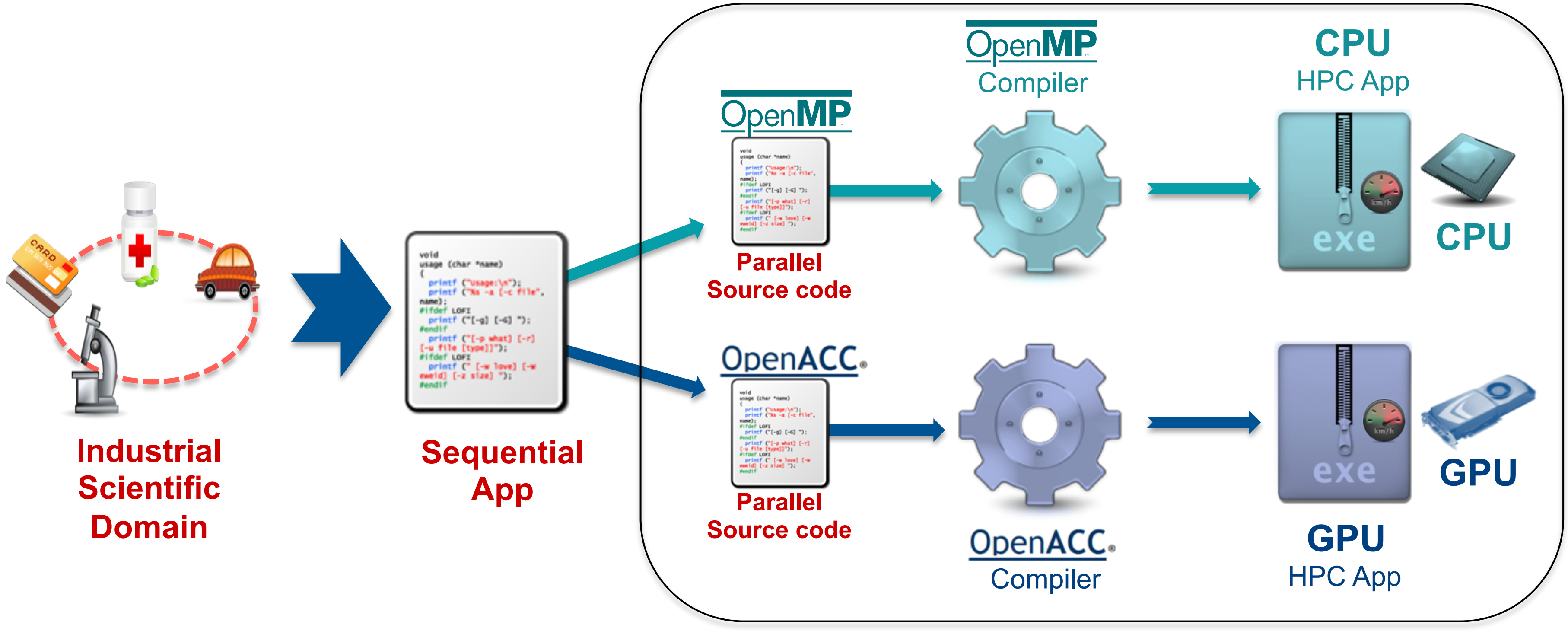
Do not try to be a Superman



Just focus on your business.

You don't need to be an expert in HPC.

Interdisciplinary Teams



Industrial Scientific Domain

Sequential App

OpenMP

Parallel Source code

OpenMP Compiler

CPU HPC App

CPU

OpenACC

Parallel Source code

OpenACC Compiler

GPU HPC App

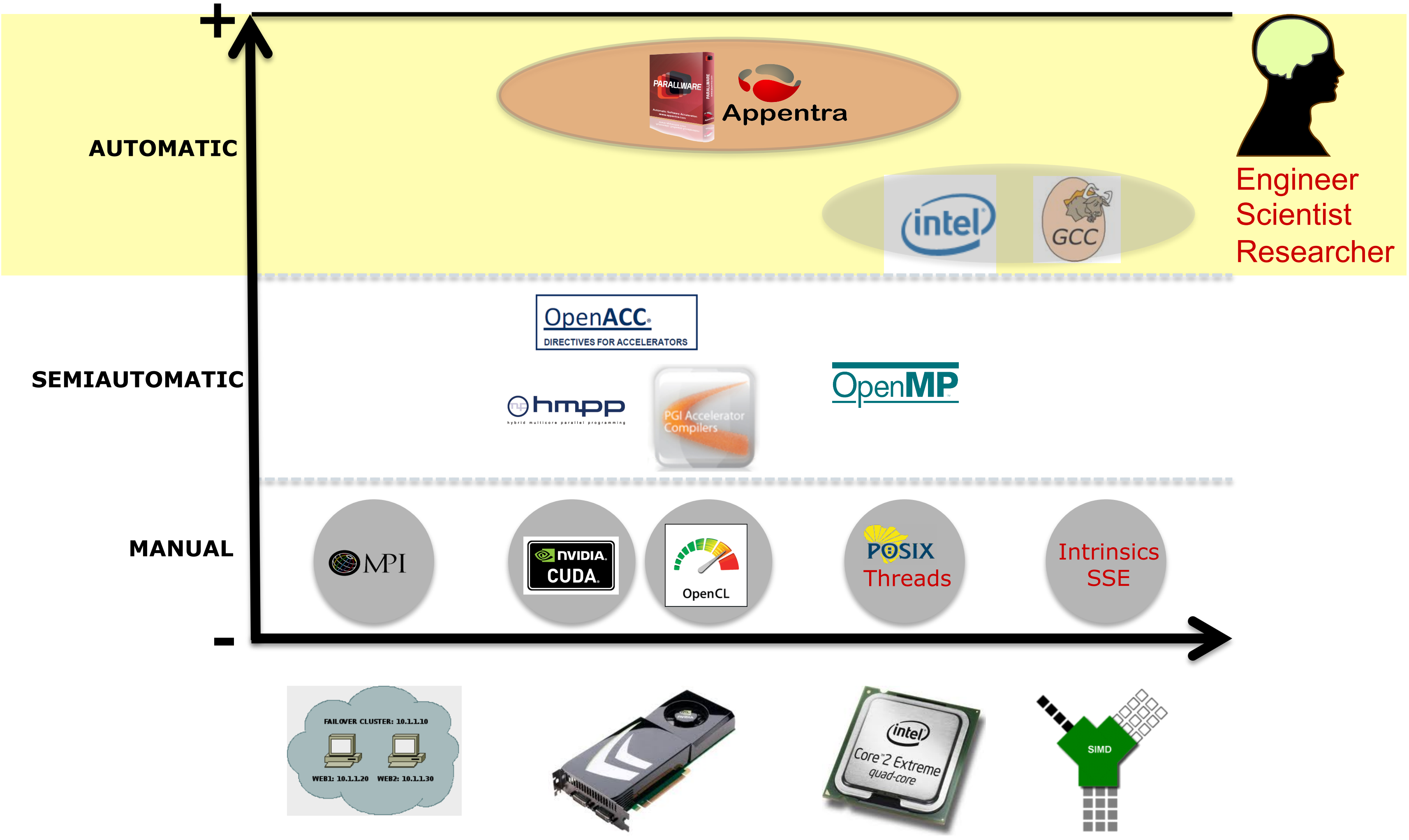
GPU

HPC App

Engineer
Scientist
Researcher

Engineer in Computer Science

Fully Automatic Tools



+

AUTOMATIC

SEMIAUTOMATIC

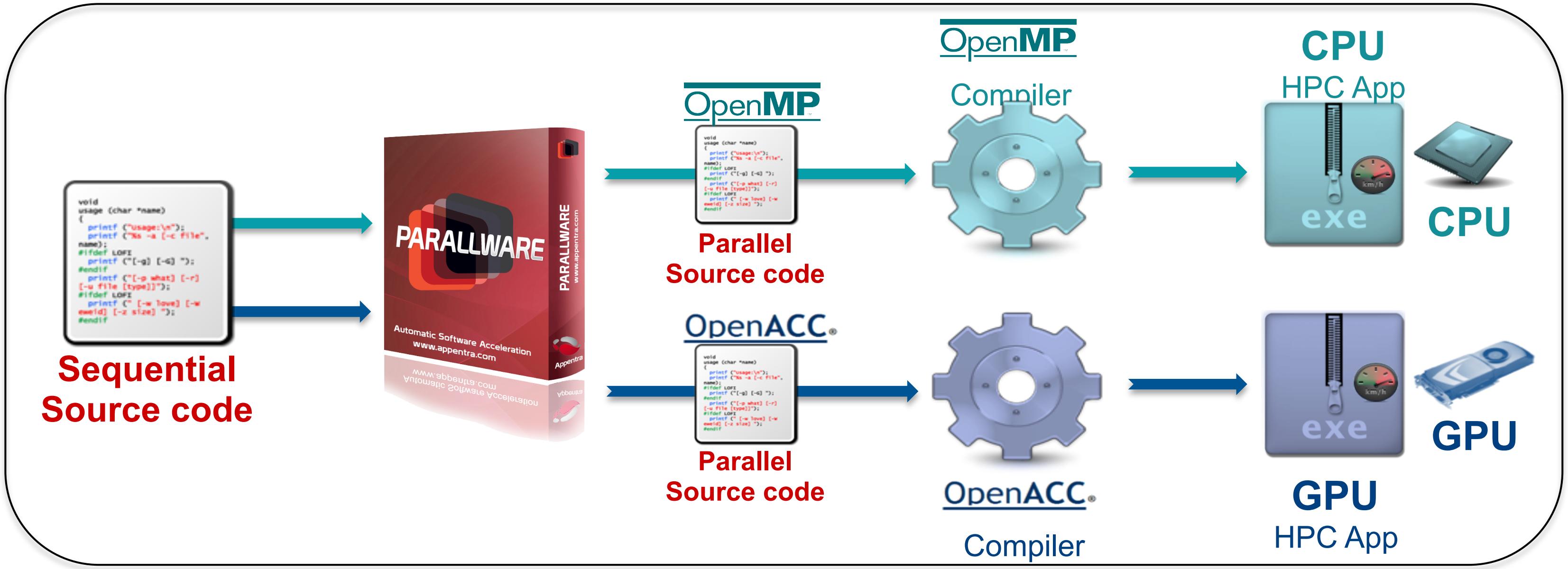
MANUAL

-

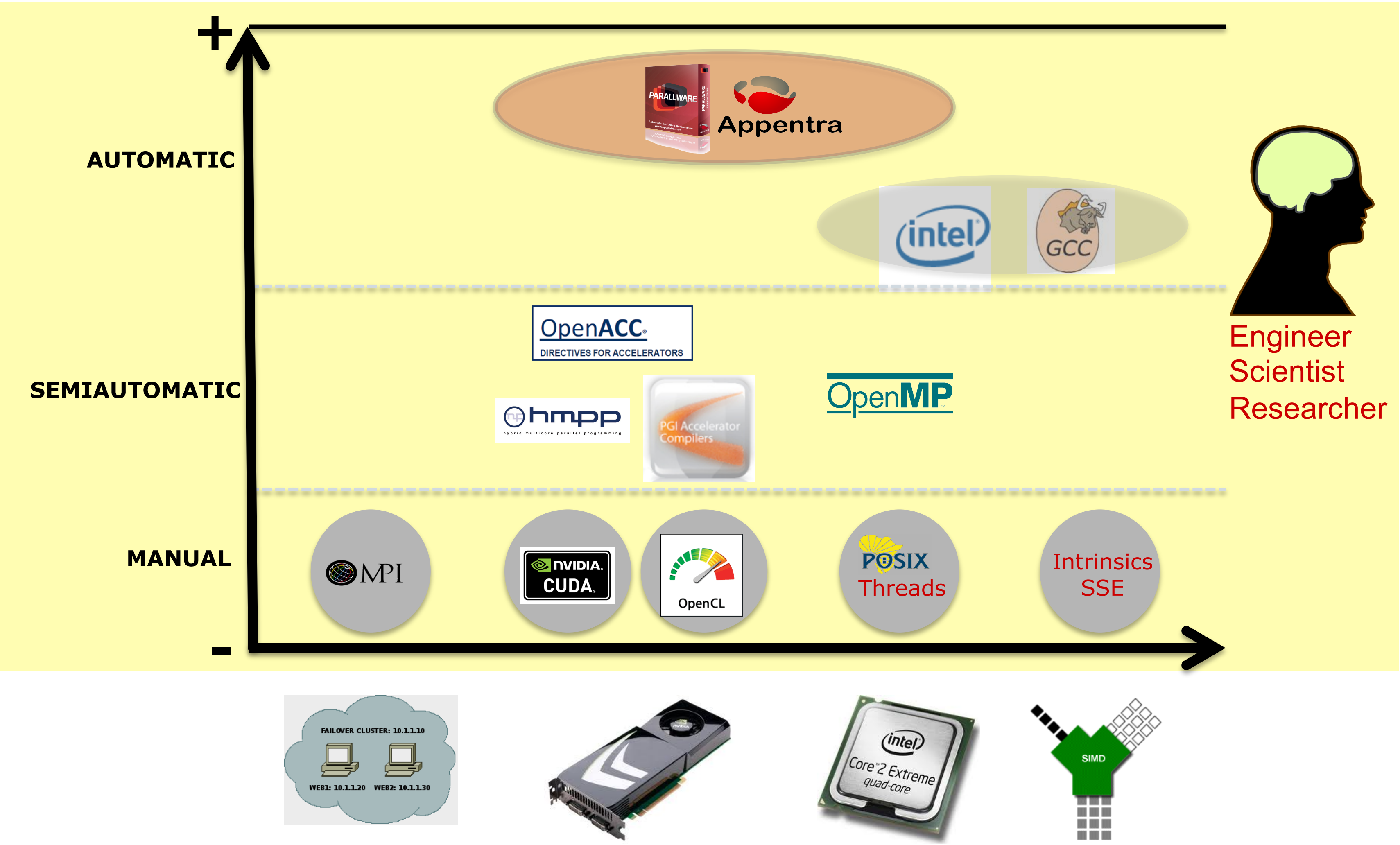
Engineer
Scientist
Researcher

ParallWare by Appentra

- Automatic parallelization of sequential Apps
- Smart source-to-source auto-parallelizer
  - **R+D+i** on advanced parallelizing compilation techniques



Software “Design Patterns”



AUTOMATIC

SEMIAUTOMATIC

MANUAL

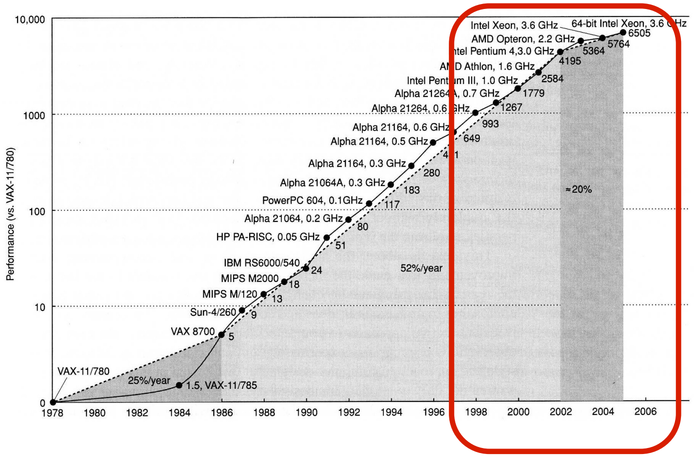
Engineer
Scientist
Researcher

How can I be more productive?

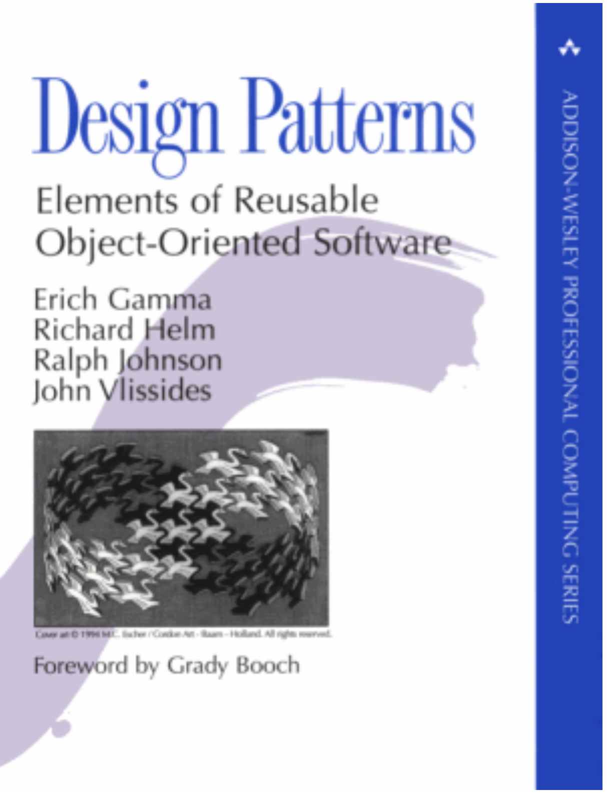
Are there “Design Patterns” to help writing HPC Apps?

Object-Oriented Design Patterns

2nd Software Crisis (80s-90s)



- Build&Maintain big&complex&robust Apps
- Object-oriented programming
- New languages & tools: C++, C#, Java
- App-oriented approach: Design patterns
- Portability at the cost of performance



Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

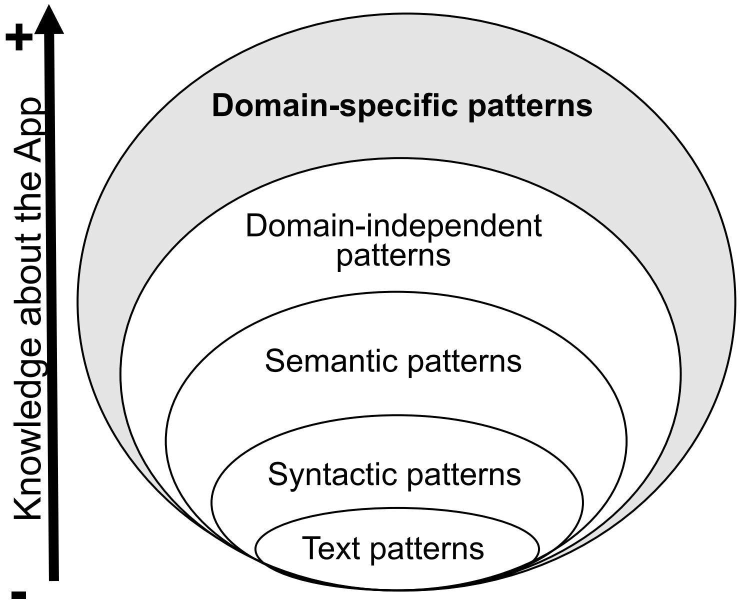
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Design Patterns for HPC

- In the literature there are many different types of design patterns for HPC Apps.
- Design patterns for parallelism discovery:
  - Task decomposition pattern
  - Data decomposition pattern
- Design patterns for the algorithm structure:
  - Task parallelism pattern
  - Divide and conquer pattern
  - Pipeline pattern
  - Event-based coordination pattern
- Design patterns for the program structure:
  - SPMD pattern
  - Master/Slave pattern
  - Shared queue pattern
- .../...

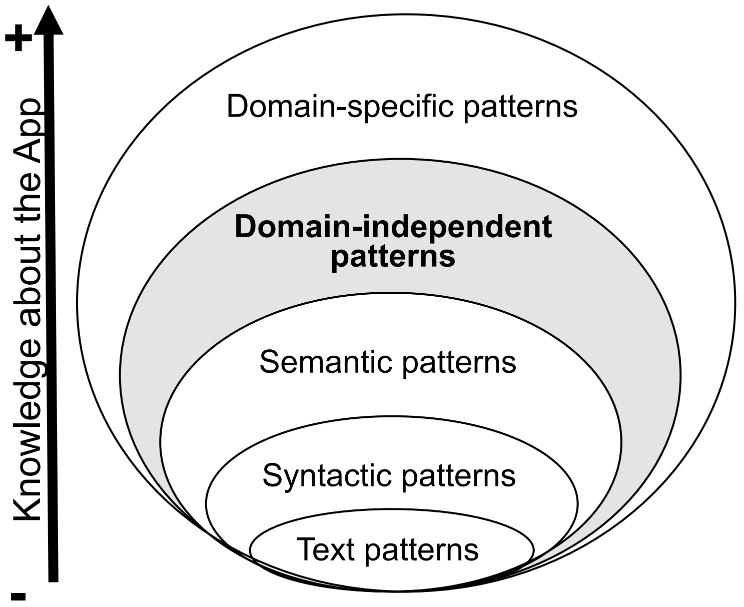
The App-Oriented Perspective

- Modern business areas increasingly rely on numerical simulation
  - E.g., automotive, aerospace, civil engineering, chemistry
- Scientists&Engineers write compute-intensive Apps that compute an approximation of the solution
- Well-known domain-specific patterns are used to avoid reinventing the wheel.
  - Fast&Optimized libraries
- Examples:
  - Matrix-Vector product
  - Vector-Vector add
  - Vector-Vector inner product
  - FFT
  - Sort
  - Compute PI
  - .../...



The App-Oriented Perspective

- Sequential Apps are written in programming languages
- Programming languages provide a limited set of building blocks
  - Programs, functions, procedures
  - Loops (e.g., for, while)
  - Conditional instructions (e.g., if-then-else, switch)
  - Processing instructions (e.g., +, *, -)



How **domain-independent** patterns look like?

The App-Oriented Perspective

```
sum = 0.0
for( i=0; i<N; i++ ) {
     sum = sum + ( 4.0 / ( 1+( (i - 0.5)/N )^2 ) )
}
pi = sum / N
```

Code: Compute an approximation of ∏ using the integration method.

```
sum = 0.0
for( i=0; i<N; i++ ) {
     sum = sum + A[i]
}
```

Code: Compute the sum of the entries of an array.

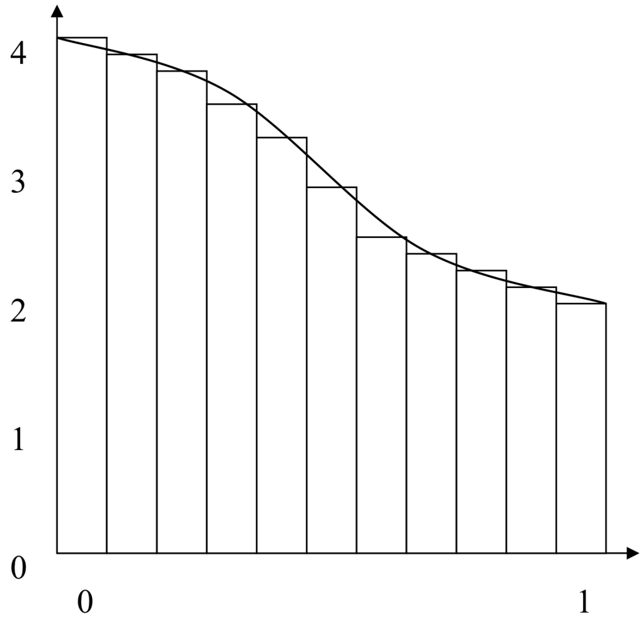
Both sequential Apps compute a **sum of values**

Are there “Design Patterns” to help writing HPC Apps?

Can you give an example of a parallel design pattern?

Case study: Algorithm ∏

- Aproximación del valor de ∏ mediante la integración de *$4/(1+x^2)$* en el intervalo [0,1].
- Dividir el intervalo en *N* subintervalos de longitud *1/N*.



- Para cada subintervalo se calcula el área del rectángulo cuya altura es el valor de $4/(1+x^2)$ en su punto medio
- La suma de las áreas de los N rectángulos aproxima el área bajo la curva
- La aproximación de ∏ es más precisa cuando *N*→∞.

Parallelization of Algorithm ∏

- Reparto del trabajo de calcular el area de *N* rectángulos entre un *P* procesadores.

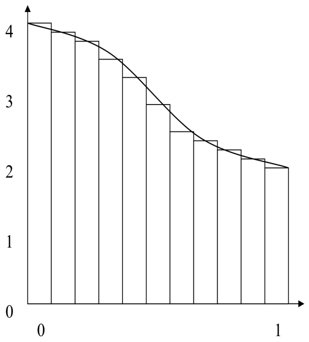
- Nº rectángulos por procesador:

  $N_p = N/P$

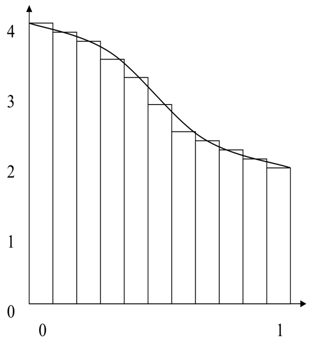
  donde p$\in$ {0,…*P*-1}

- Estimación de ∏ :

  $(S_0 + \ldots + S_{p-1})/N$



…



p=0

⇨ S_0

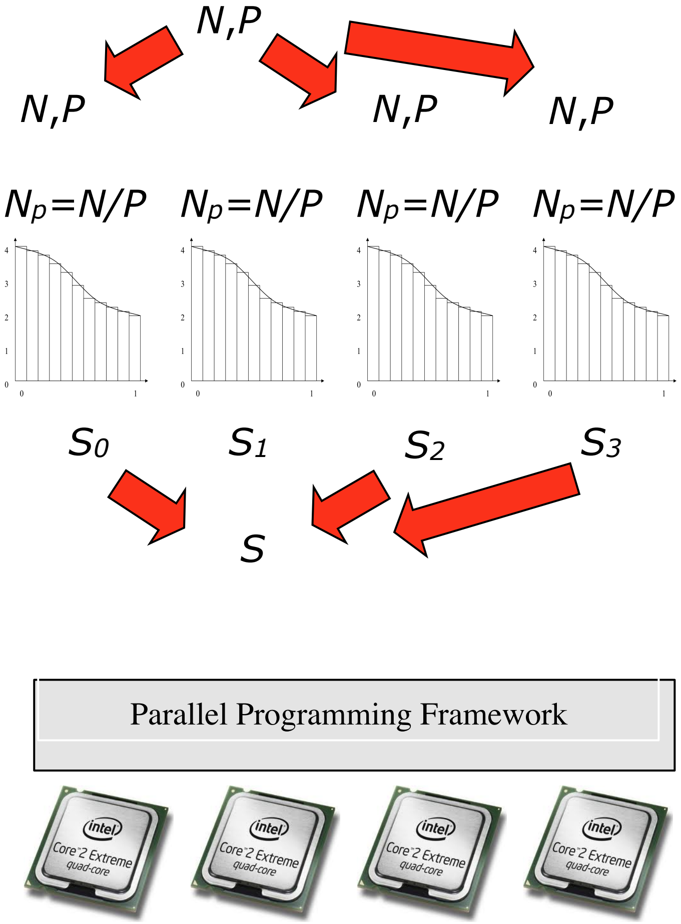
p=P-1

⇨ S_{P-1}

Parallelization of Algorithm Π

- Stage 1:
  - Broadcast *N*, *P*
- Stage 2:
  - Distribute loop iterations
  - Compute partial sums S_p at each processor
- Stage 3:
  - Gather all partial sums
  - Compute global sum

$$S = S_0 + \ldots + S_{p-1}$$



N,P

N,P N,P N,P

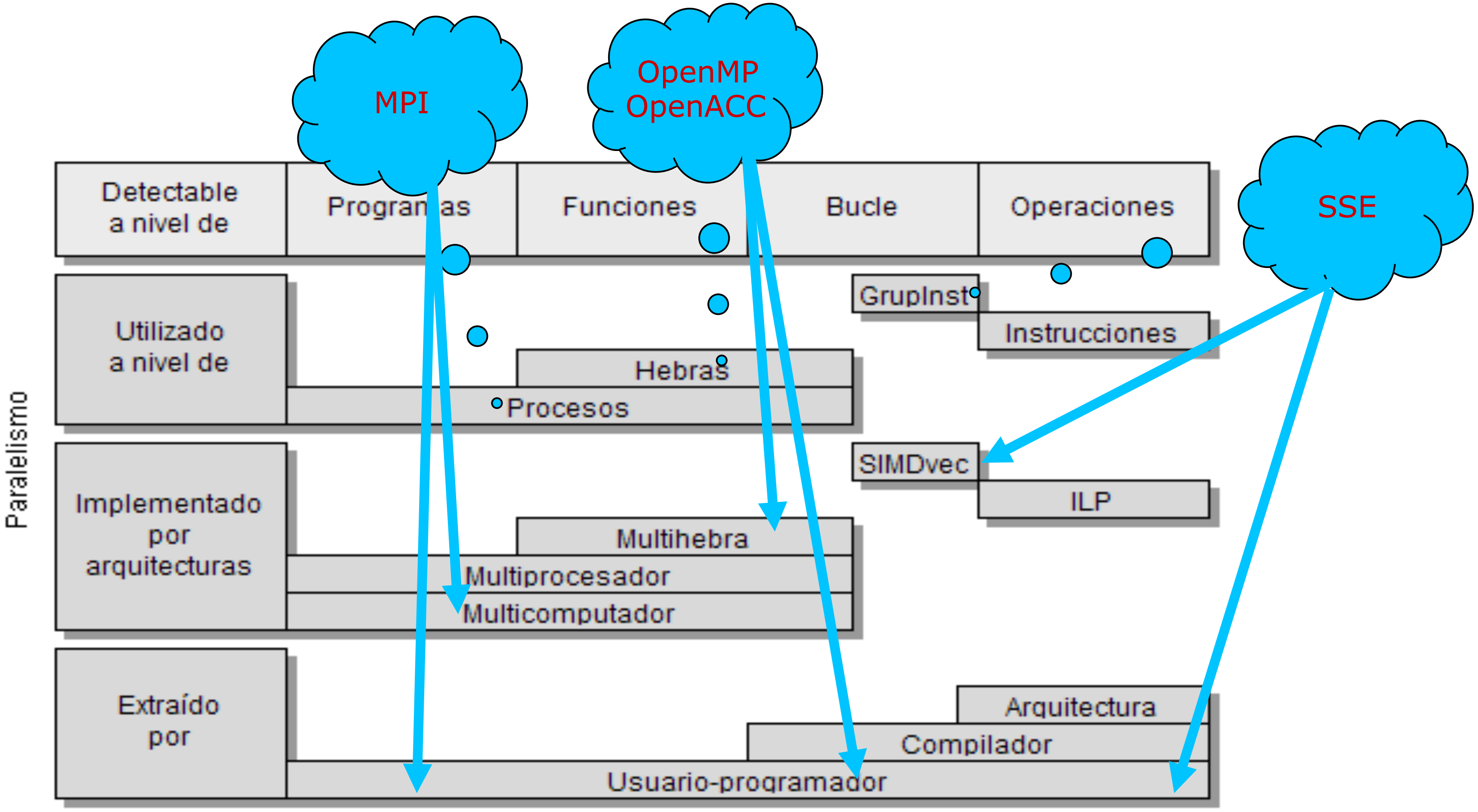
$N_p = N/P$ $N_p = N/P$ $N_p = N/P$ $N_p = N/P$

S_0 S_1 S_2 S_3

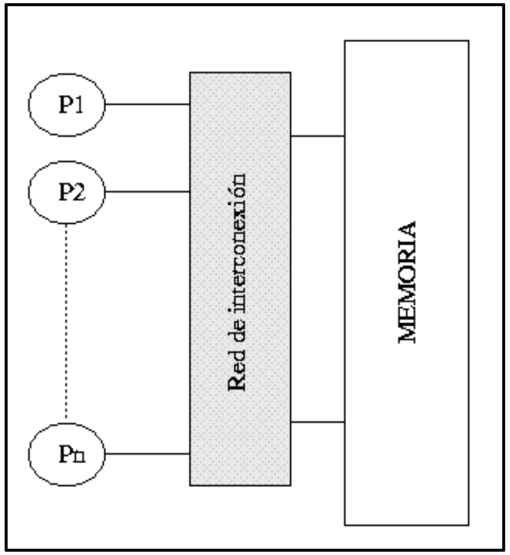
S

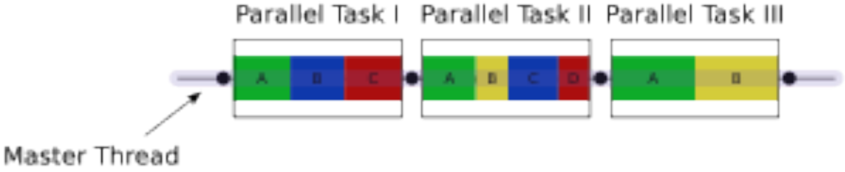
Parallel Programming Framework

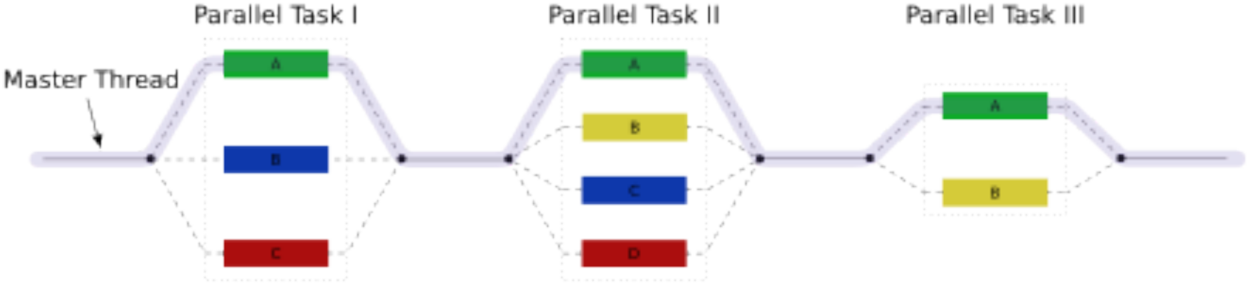
Parallel Implementations of Π



OpenMP Execution Model







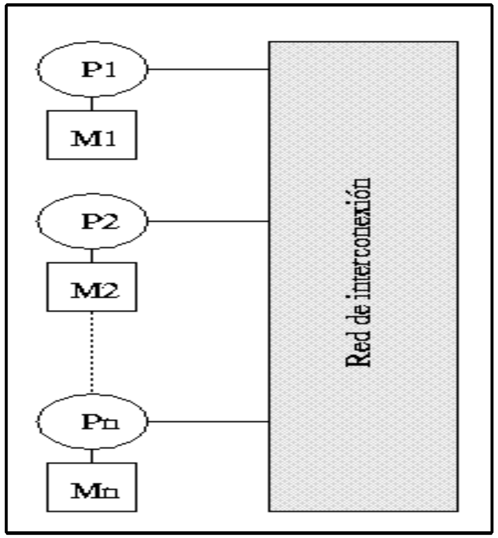
OpenMP

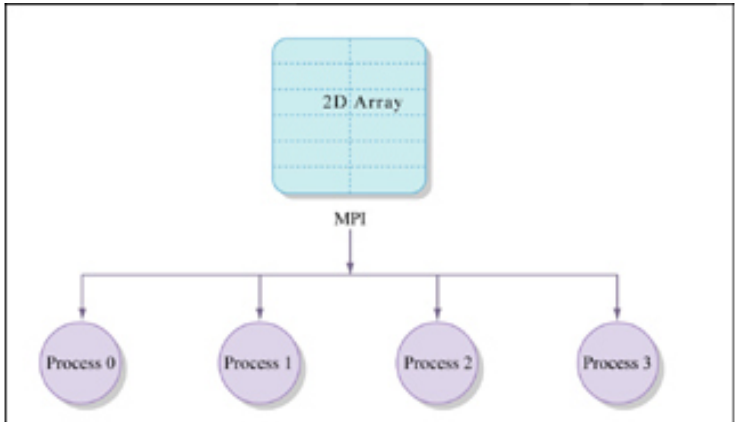
```
sum = 0.0
#pragma omp parallel for reduction(+:sum)
for( i=0; i<N; i++ ) {
      sum = sum + ( 4.0 / ( 1+( (i - 0.5)/N )^2 ) )
}
/* Cálculo de PI */
pi = sum / N
```

```
sum = 0.0
#pragma omp parallel shared (N) private(i)\
                                    reduction(+:sum)
{
#pragma for schedule(static)
for( i=0; i<N; i++ ) {
      sum = sum + ( 4.0 / ( 1+( (i - 0.5)/N )^2 ) )
}
}

/* Cálculo de PI */
pi = sum / N
```

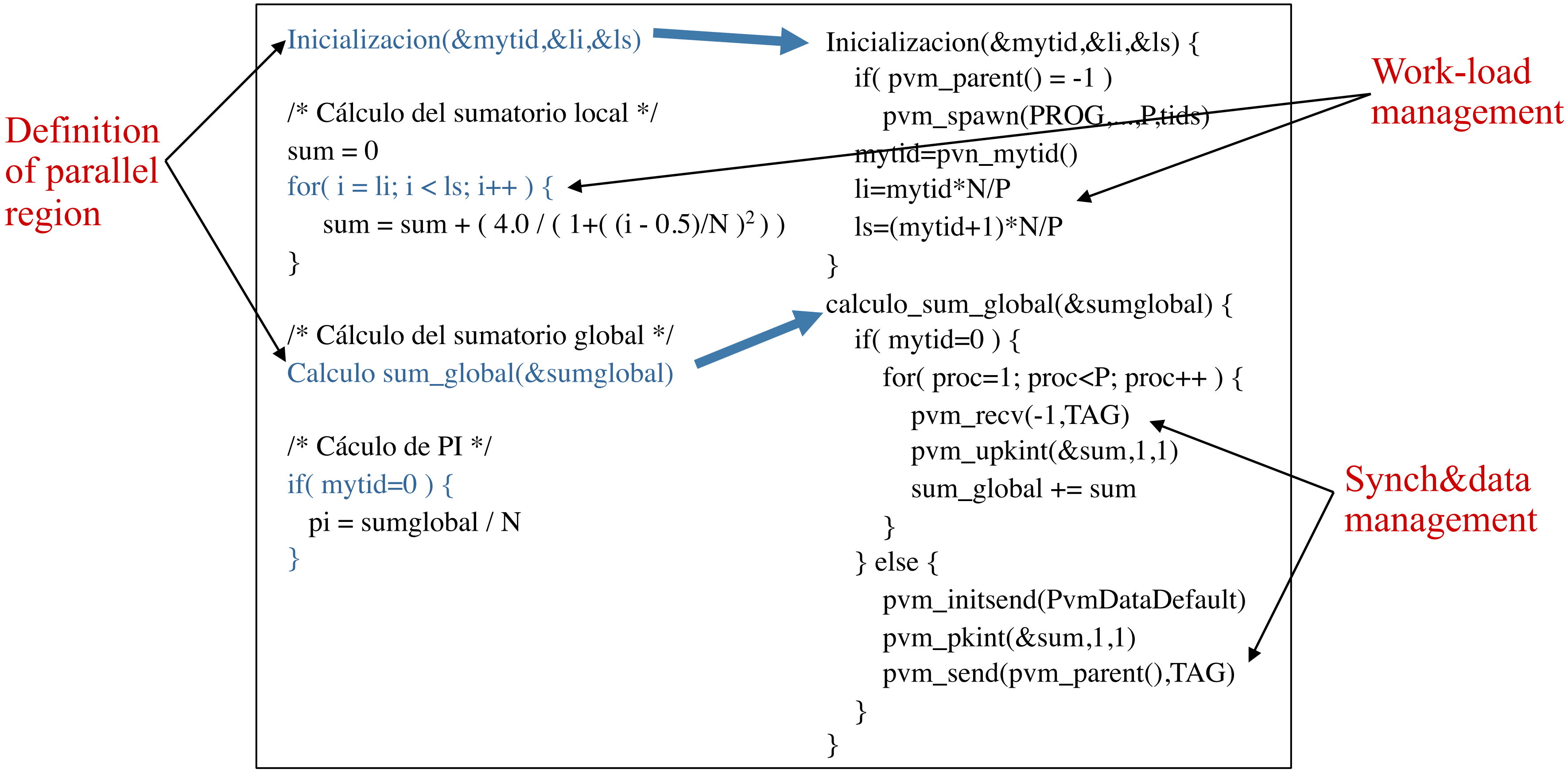
```
sum = 0.0
#pragma omp parallel shared (N) private(i)\
                                    private(sum_aux)
{
     sum_aux = 0;
     #pragma for schedule(static)
     for( i=0; i<N; i++ ) {
         sum_aux = sum_aux + ( 4.0 / ( 1+( (i - 0.5)/N )^2 ) )
     }
     #pragma atomic
     sum = sum + sum_aux;
}

/* Cálculo de PI */
pi = sum / N
```

MPI Execution Model





MPI



Definition of parallel region

```
Inicializacion(&mytid,&li,&ls)

/* Cálculo del sumatorio local */
sum = 0
for( i = li; i < ls; i++ ) {
    sum = sum + ( 4.0 / ( 1+( (i - 0.5)/N )^2 ) )
}

/* Cálculo del sumatorio global */
Calculo sum_global(&sumglobal)

/* Cáculo de PI */
if( mytid=0 ) {
  pi = sumglobal / N
}
```

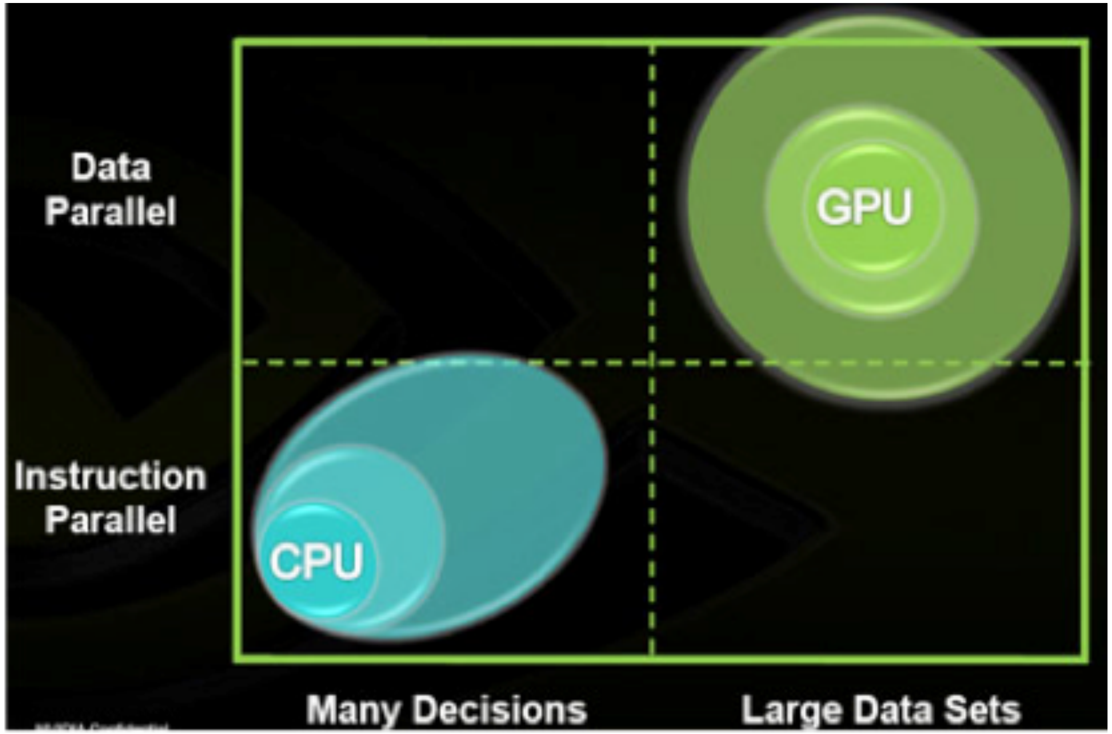
```
Inicializacion(&mytid,&li,&ls) {
   if( pvm_parent() = -1 )
      pvm_spawn(PROG,...,P,tids)
   mytid=pvn_mytid()
   li=mytid*N/P
   ls=(mytid+1)*N/P
}
calculo_sum_global(&sumglobal) {
   if( mytid=0 ) {
      for( proc=1; proc<P; proc++ ) {
         pvm_recv(-1,TAG)
         pvm_upkint(&sum,1,1)
         sum_global += sum
      }
   } else {
      pvm_initsend(PvmDataDefault)
      pvm_pkint(&sum,1,1)
      pvm_send(pvm_parent(),TAG)
   }
}
```

Work-load management

Synch&data management

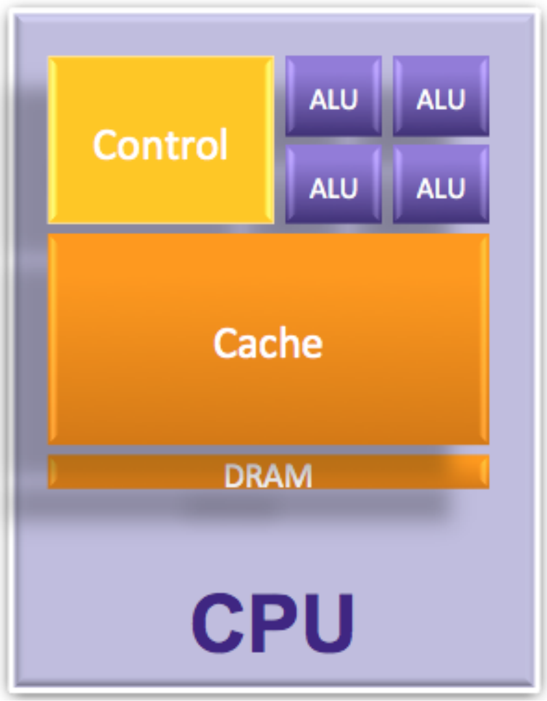
Code: PVM implementation with block/consecutive work-sharing.

GPU Design Goals

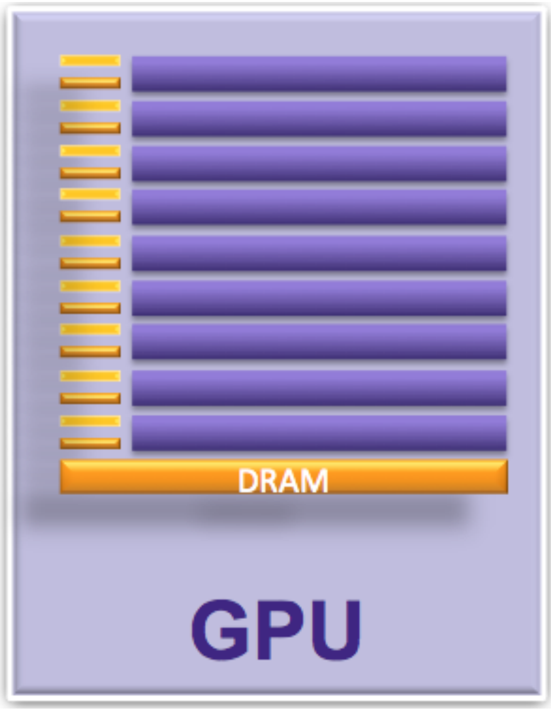


Source: http://www.legitreviews.com/article/1100/1/

GPU Architecture



General purpose architecture

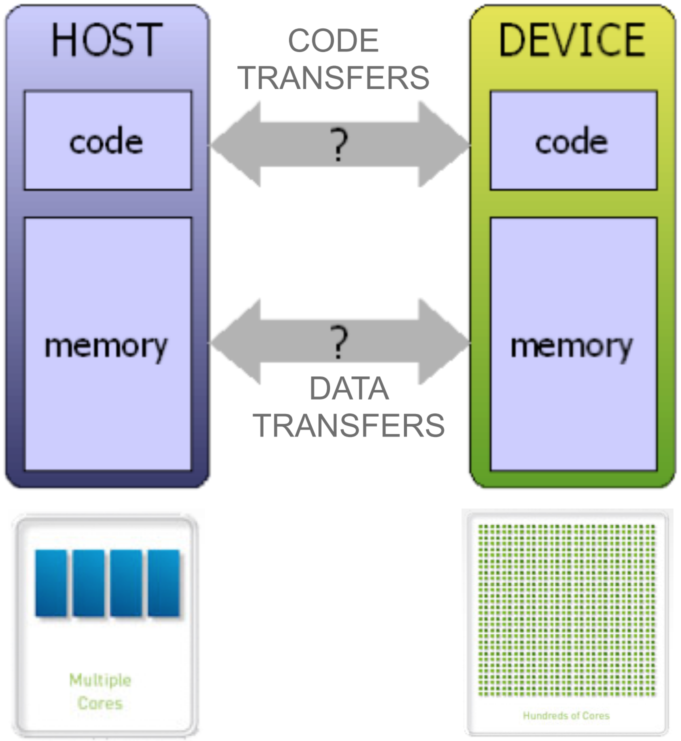


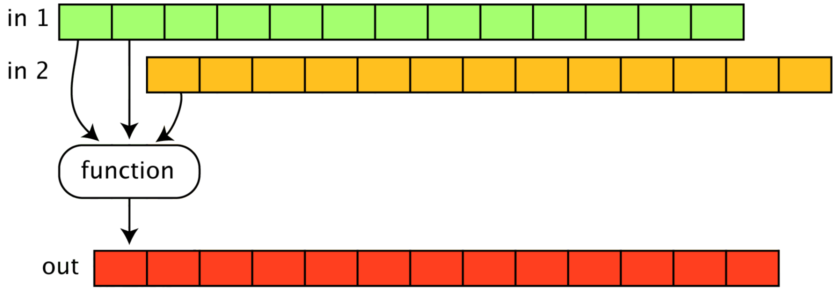
Massively data parallel

Needs 1000s of computation threads to be efficient

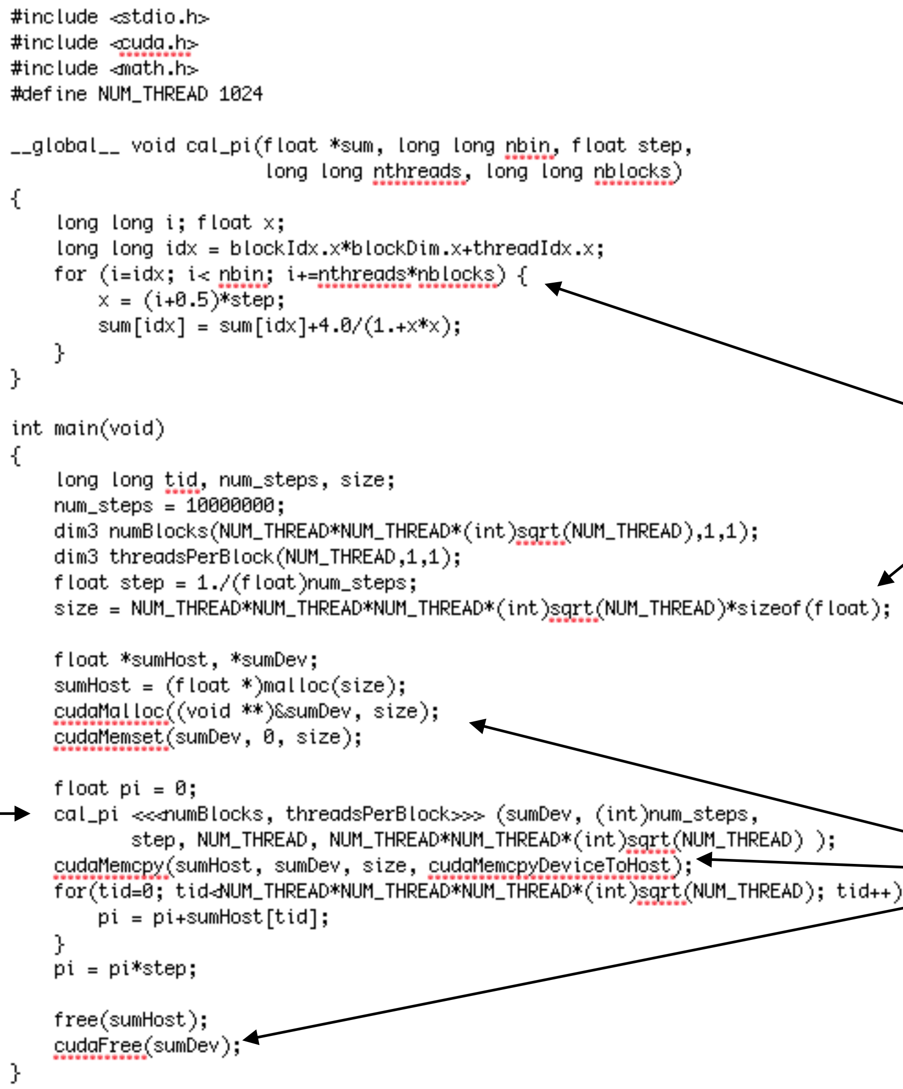
GPU Execution Model

- Host-driven execution:
  - Allocate memory space on the accelerator
  - Initiate data transfers
  - Launch computations (streaming model)
  - Wait for completion
  - Deallocate memory space





CUDA



```
#include <stdio.h>
#include <cuda.h>
#include <math.h>
#define NUM_THREAD 1024

__global__ void cal_pi(float *sum, long long nbin, float step,
                       long long nthreads, long long nblocks)
{
    long long i; float x;
    long long idx = blockIdx.x*blockDim.x+threadIdx.x;
    for (i=idx; i< nbin; i+=nthreads*nblocks) {
        x = (i+0.5)*step;
        sum[idx] = sum[idx]+4.0/(1.+x*x);
    }
}

int main(void)
{
    long long tid, num_steps, size;
    num_steps = 10000000;
    dim3 numBlocks(NUM_THREAD*NUM_THREAD*(int)sqrt(NUM_THREAD),1,1);
    dim3 threadsPerBlock(NUM_THREAD,1,1);
    float step = 1./(float)num_steps;
    size = NUM_THREAD*NUM_THREAD*NUM_THREAD*(int)sqrt(NUM_THREAD)*sizeof(float);

    float *sumHost, *sumDev;
    sumHost = (float *)malloc(size);
    cudaMalloc((void **)&sumDev, size);
    cudaMemset(sumDev, 0, size);

    float pi = 0;
    cal_pi <<<numBlocks, threadsPerBlock>>> (sumDev, (int)num_steps,
           step, NUM_THREAD, NUM_THREAD*NUM_THREAD*(int)sqrt(NUM_THREAD) );
    cudaMemcpy(sumHost, sumDev, size, cudaMemcpyDeviceToHost);
    for(tid=0; tid<NUM_THREAD*NUM_THREAD*NUM_THREAD*(int)sqrt(NUM_THREAD); tid++) {
        pi = pi+sumHost[tid];
    }
    pi = pi*step;

    free(sumHost);
    cudaFree(sumDev);
}
```

Work-load management

Definition of parallel region

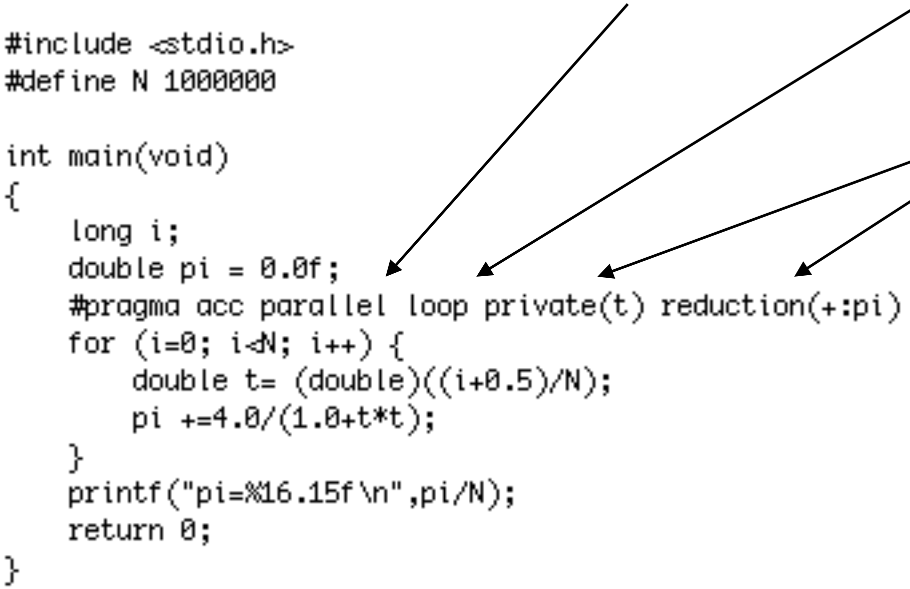
Synch&data management

OpenACC

Definition of parallel region

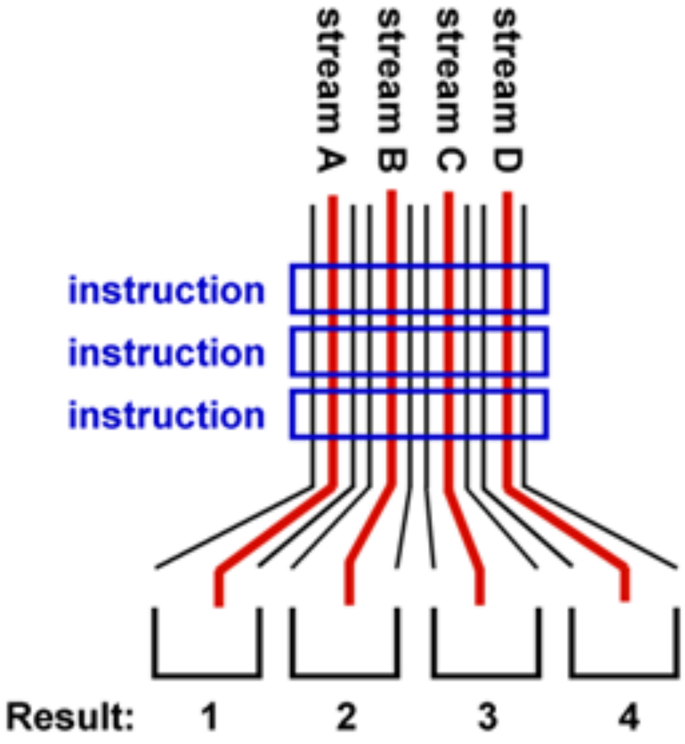
Work-load management

Synch and data management

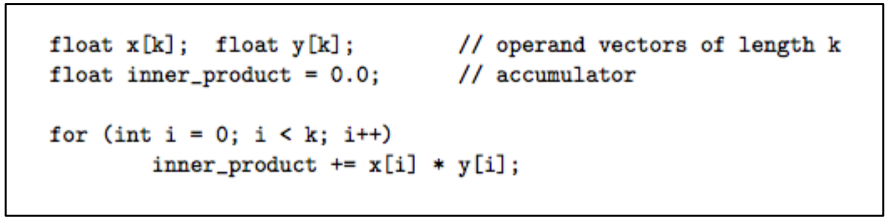


```
#include <stdio.h>
#define N 1000000

int main(void)
{
    long i;
    double pi = 0.0f;
    #pragma acc parallel loop private(t) reduction(+:pi)
    for (i=0; i<N; i++) {
        double t= (double)((i+0.5)/N);
        pi +=4.0/(1.0+t*t);
    }
    printf("pi=%16.15f\n",pi/N);
    return 0;
}
```

SSE Execution Model

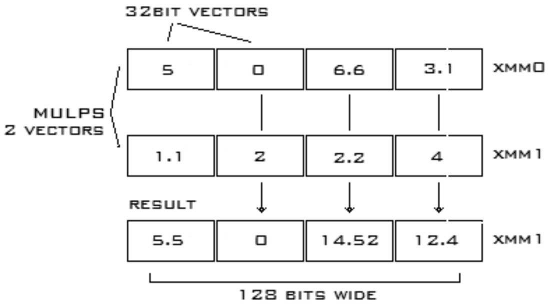


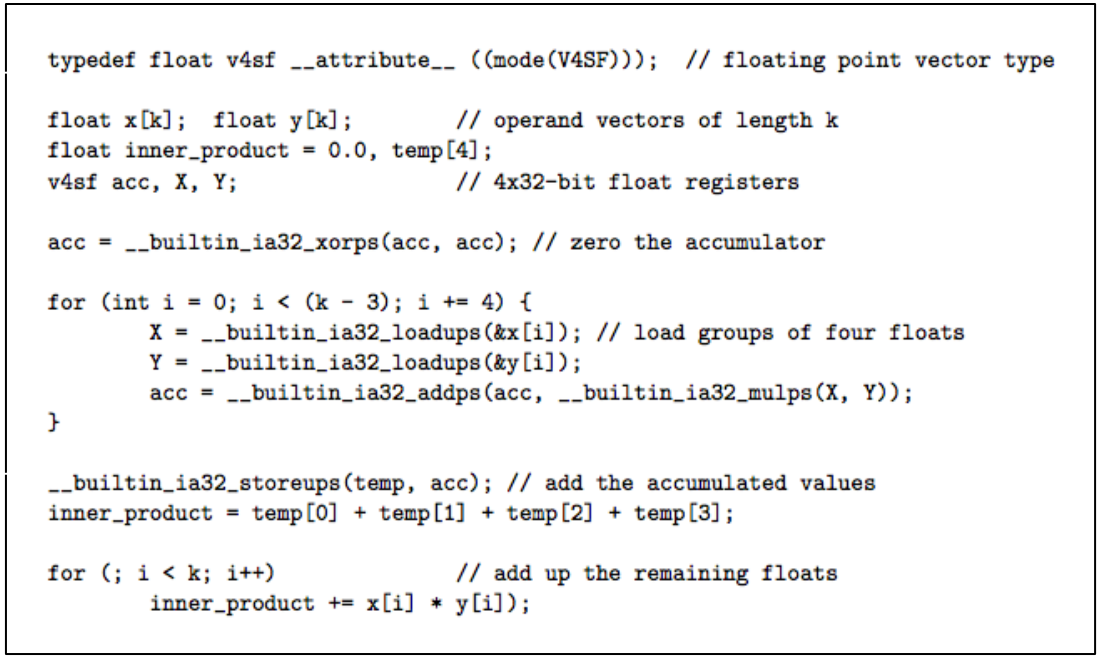
SSE (SIMDization/Vectorization)



```
float x[k];  float y[k];         // operand vectors of length k
float inner_product = 0.0;       // accumulator

for (int i = 0; i < k; i++)
        inner_product += x[i] * y[i];
```

Code: Inner product of two vectors.

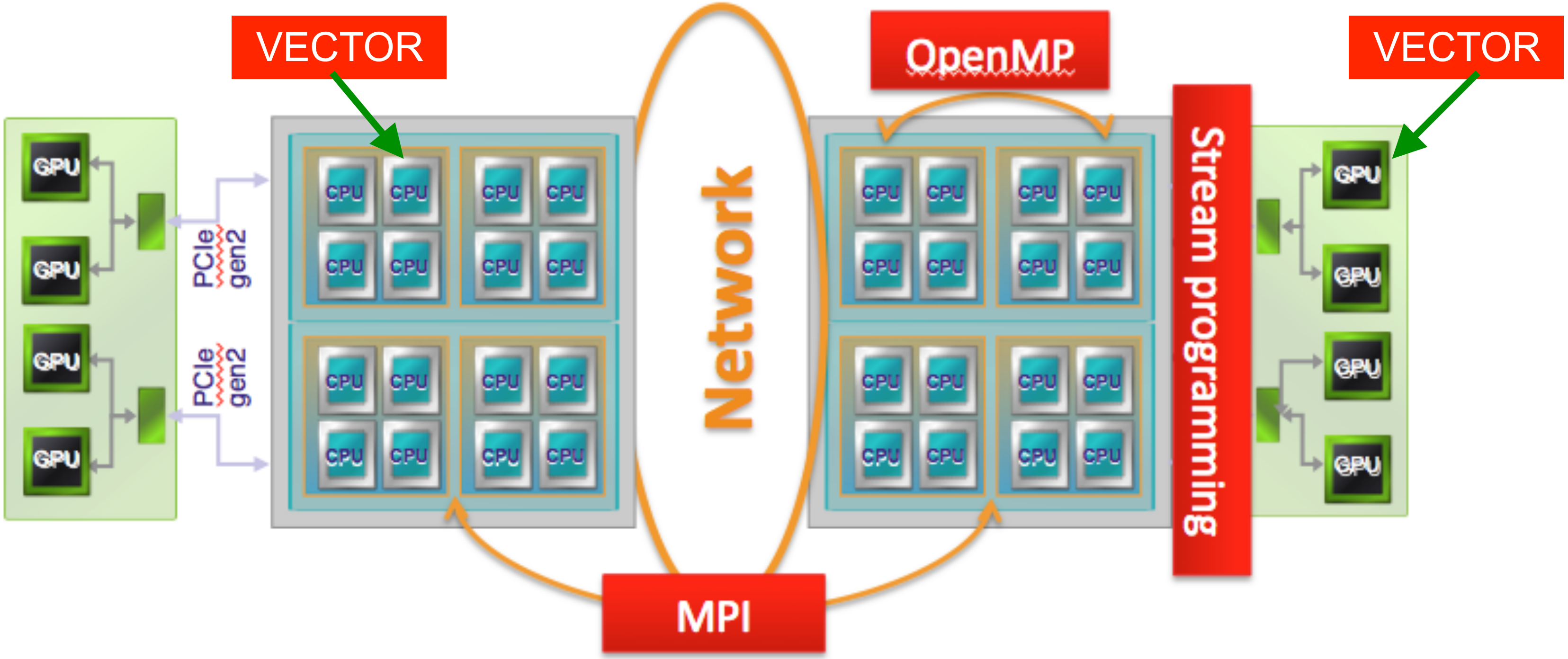




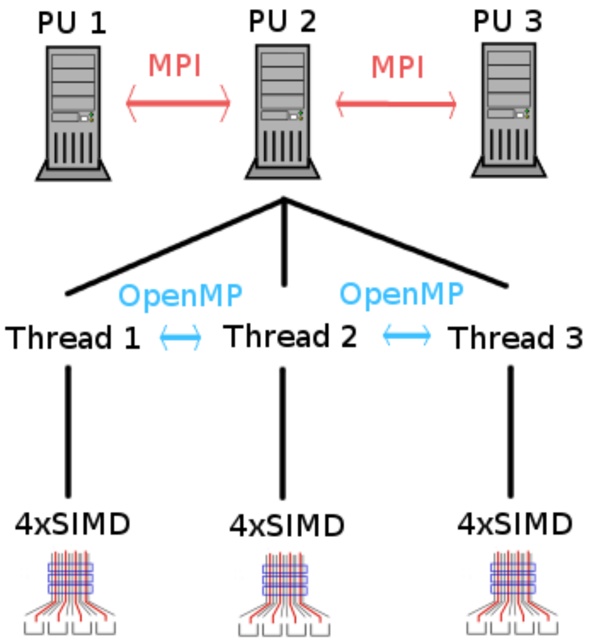
```
typedef float v4sf __attribute__ ((mode(V4SF)));  // floating point vector type

float x[k];  float y[k];         // operand vectors of length k
float inner_product = 0.0, temp[4];
v4sf acc, X, Y;                  // 4x32-bit float registers

acc = __builtin_ia32_xorps(acc, acc); // zero the accumulator

for (int i = 0; i < (k - 3); i += 4) {
        X = __builtin_ia32_loadups(&x[i]); // load groups of four floats
        Y = __builtin_ia32_loadups(&y[i]);
        acc = __builtin_ia32_addps(acc, __builtin_ia32_mulps(X, Y));
}

__builtin_ia32_storeups(temp, acc); // add the accumulated values
inner_product = temp[0] + temp[1] + temp[2] + temp[3];

for (; i < k; i++)               // add up the remaining floats
        inner_product += x[i] * y[i]);
```

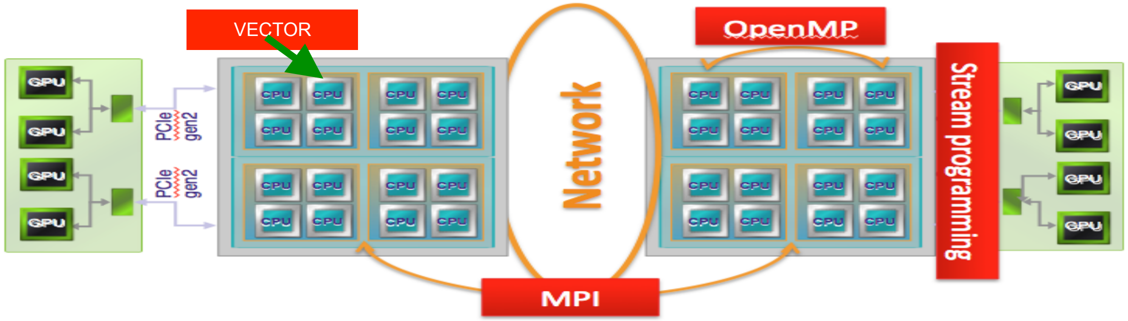
Code: SSE implementation of the inner product of two vectors.

Joint Execution Model



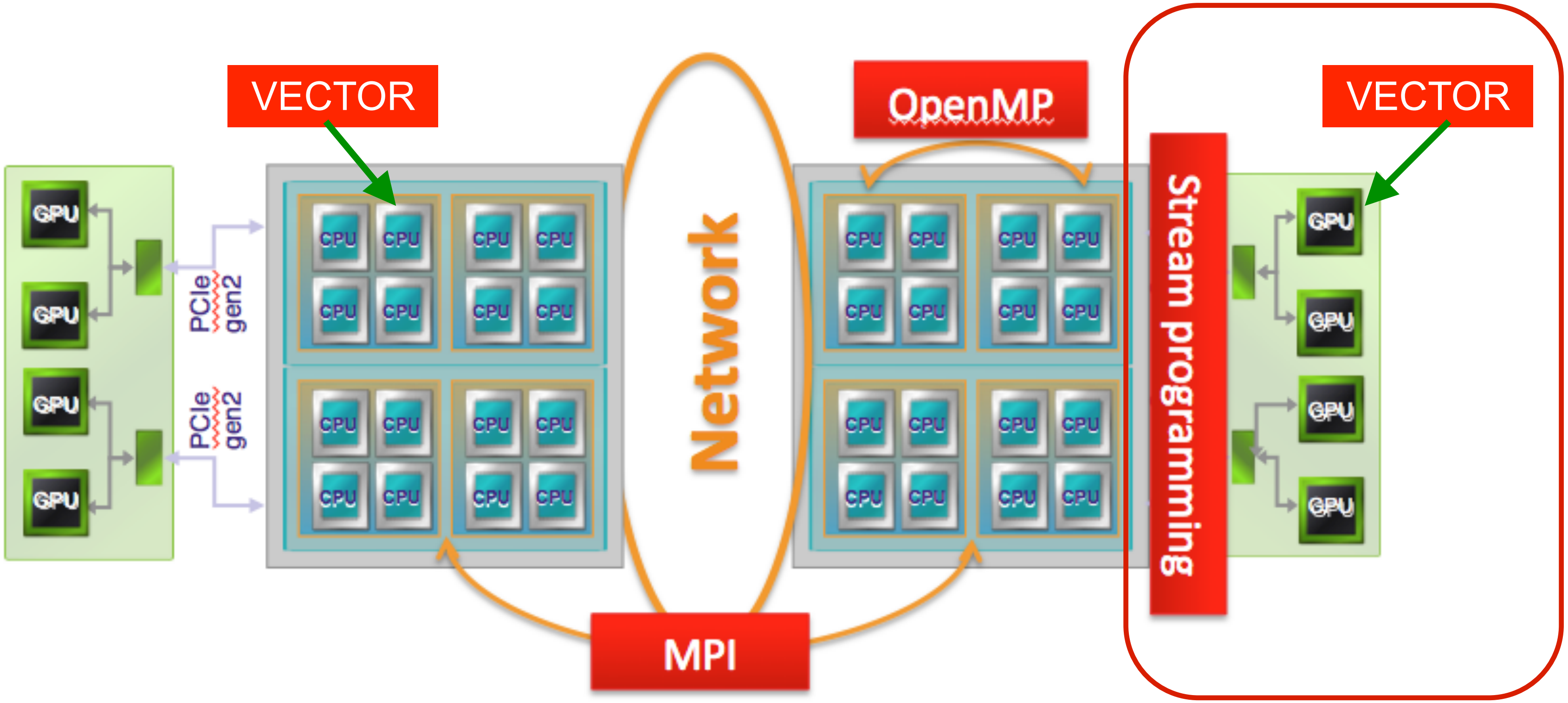
Joint Execution Model



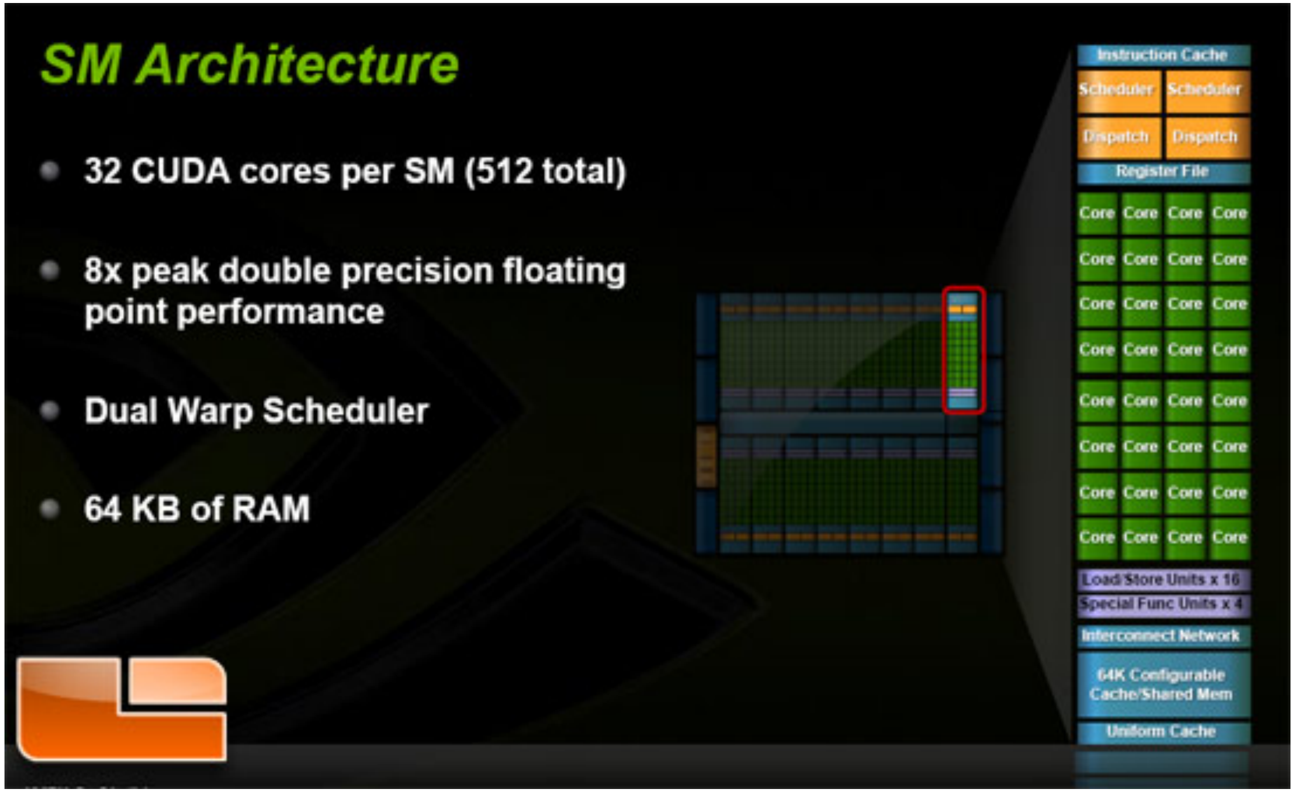


GPU Levels of Parallelism

- □ Clusters expose multiple levels of parallelism…
- □ And GPUs also expose multiple levels of parallelism!



GPU Levels of Parallelism



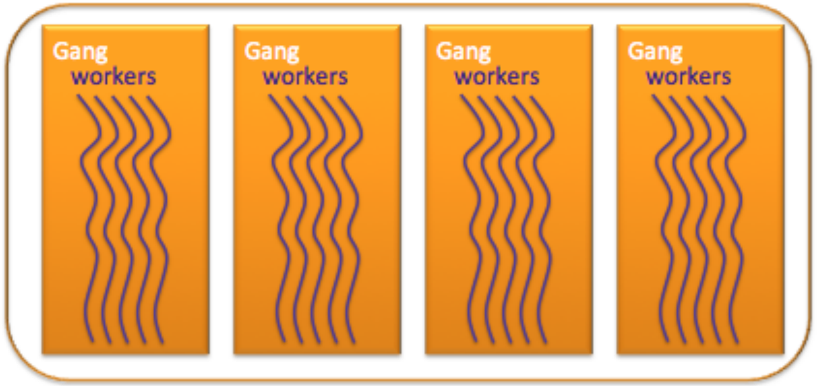
SM Architecture

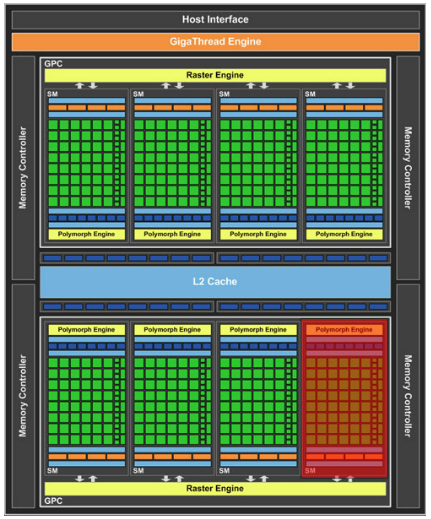
- 32 CUDA cores per SM (512 total)
- 8x peak double precision floating point performance
- Dual Warp Scheduler
- 64 KB of RAM

Fuente: http://www.legitreviews.com/article/1100/1/

GPU Levels of Parallelism

- □ Coarse-grain: **gangs**
- □ Fine-grain: **workers**
- □ Finest-grain: **vector**

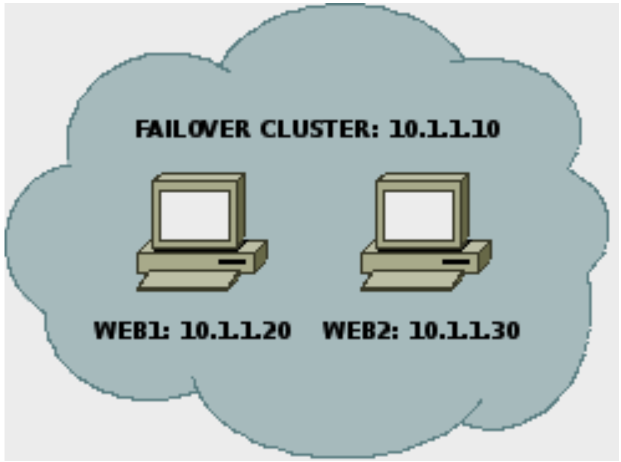




GPU internal architecture

#Blocks
#Threads/Block
#WarpThreads

#Processes
#Threads/Process
Instructions SSE



Cluster of PCs

Can you give an example of a parallel design pattern?

Are there Hw-independent key concepts?

Race Conditions

- Race conditions are programmer's nightmare
- Make the result of your parallel code unpredictable.

- What are "race conditions"?
- How can we handle "race conditions"?

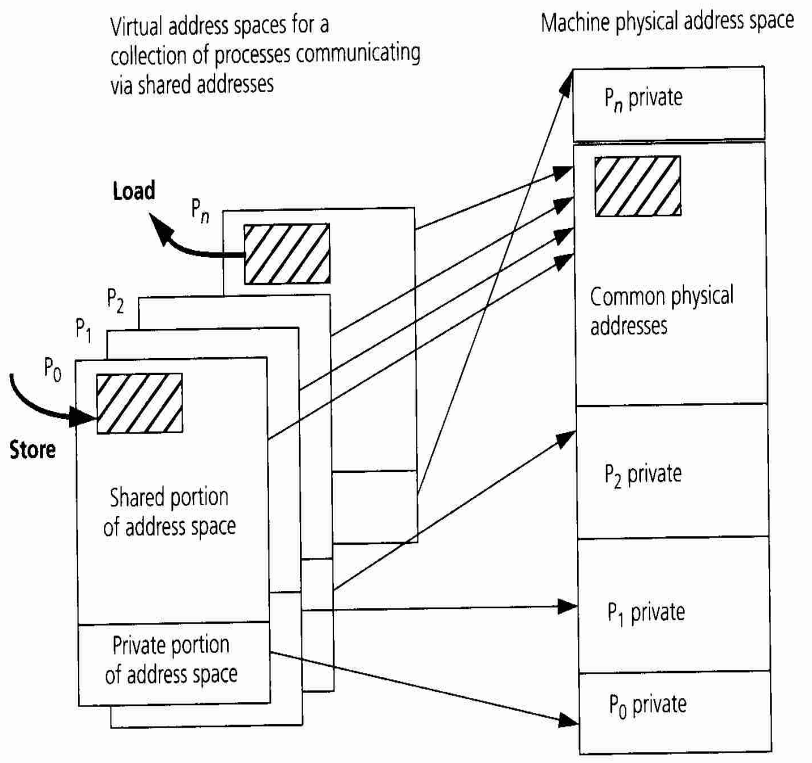

Virtual address spaces for a collection of processes communicating via shared addresses
Machine physical address space
Load
P_n
P_2
P_1
P_0
Store
Shared portion of address space
Private portion of address space
P_n private
Common physical addresses
P_2 private
P_1 private
P_0 private

FIGURE 1.14 Typical memory model for shared memory parallel programs. Collections of processes have a common region of physical addresses mapped into their virtual address space, in addition to the private region, which typically contains the stack and private data.

Race Conditions

What is the value of variable “x” at the end of the parallel region?

```
x=0;
#pragma acc parallel
{
   #pragma acc loop gangs
   for (int i=0; i<N; i++ ) {
       x = x + 1;
   }
}
```

Race Conditions

What is the value of variable “x” at the end of the parallel region?

```
x=0;
#pragma acc parallel
{
   #pragma acc loop gangs
   for (int i=0; i<N; i++ ) {
       x = x + 1;
   }
}
```

Scenario 1:

Thread0 (“x=x+1”) finishes before Thread1 begins its computation (“x=x+1”) and the value is “x=2”

Thread0: r1=0+1
Thread0: x=r1
Thread1: r2=1+1
Thread1: x=r2

Race Conditions

What is the value of variable "x" at the end of the parallel region?

```
x=0;
#pragma acc parallel
{
   #pragma acc loop gangs
   for (int i=0; i<N; i++ ) {
      x = x + 1;
   }
}
```

Scenario 1:

Thread0 ("x=x+1") finishes before Thread1 begins its computation ("x=x+1") and the value is "x=2"

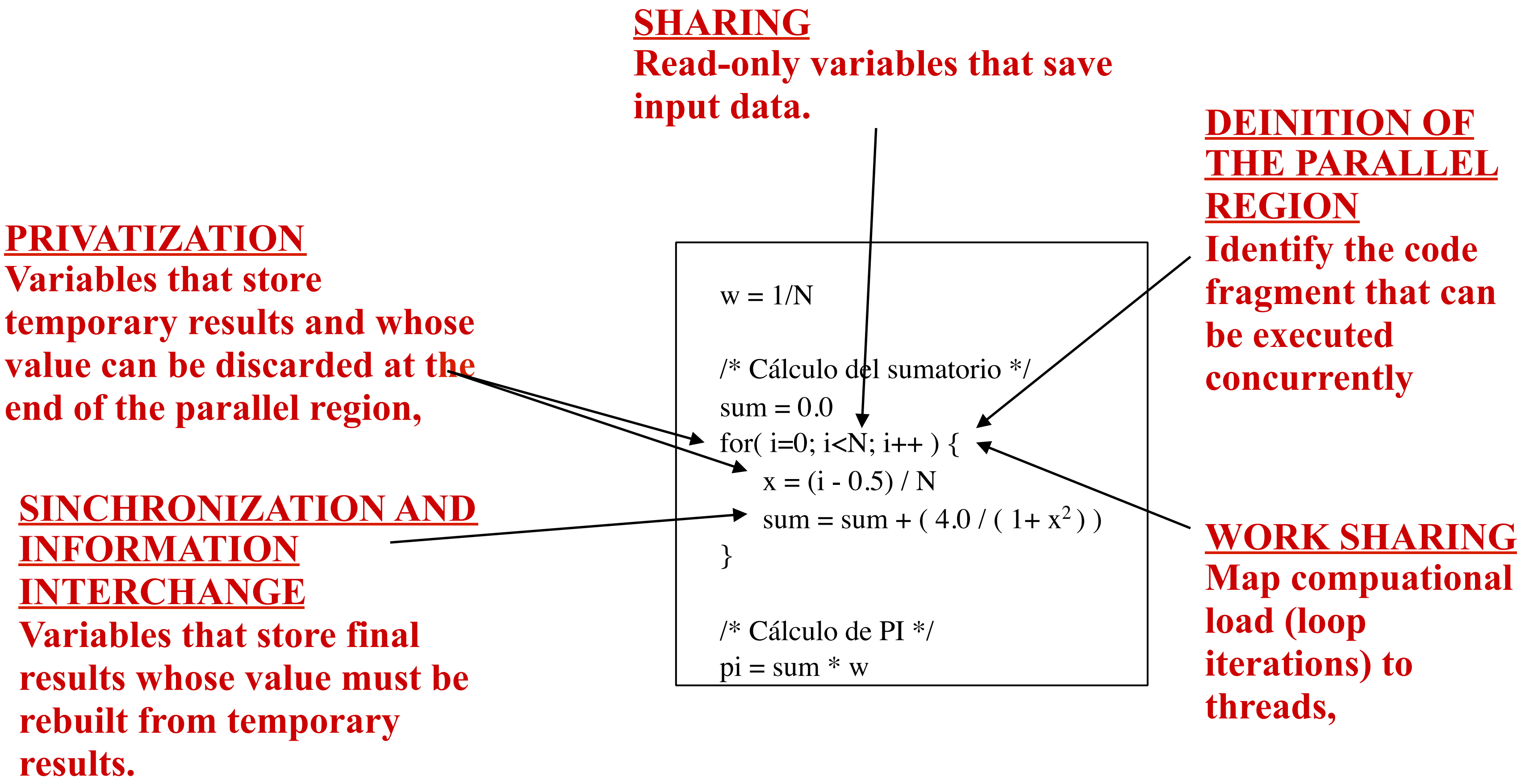
Thread0: r1=0+1
Thread0: x=r1
Thread1: r2=1+1
Thread1: x=r2

Scenario 2:

Thread0 ("x=x+1") does not finish before Thread1 begins ("x=x+1") and the value is "x=1"

Thread0: r1=0+1
Thread0: x=r1
Thread1: r2=0+1
Thread1: x=r2

An App-oriented approach



SHARING
Read-only variables that save input data.

PRIVATIZATION
Variables that store temporary results and whose value can be discarded at the end of the parallel region,

SINCHRONIZATION AND INFORMATION INTERCHANGE
Variables that store final results whose value must be rebuilt from temporary results.

```
w = 1/N

/* Cálculo del sumatorio */
sum = 0.0
for( i=0; i<N; i++ ) {
    x = (i - 0.5) / N
    sum = sum + ( 4.0 / ( 1+ x^2 ) )
}

/* Cálculo de PI */
pi = sum * w
```

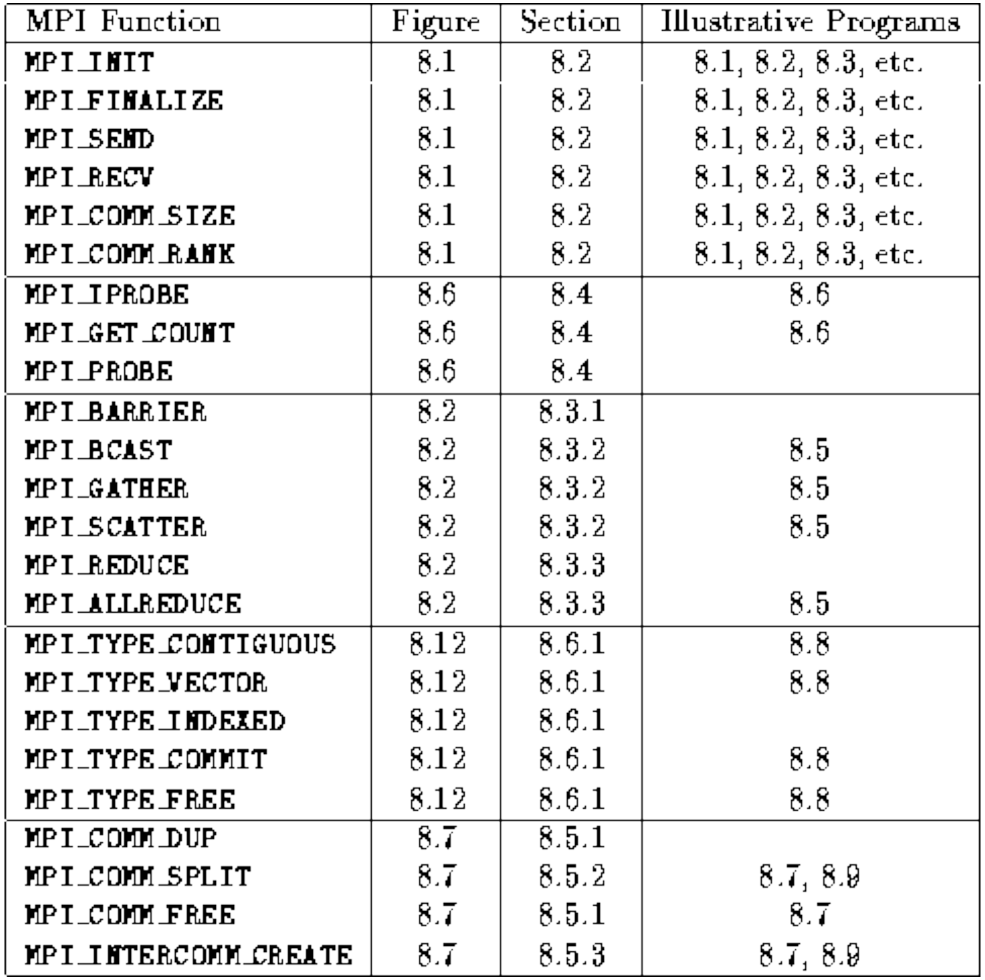
DEINITION OF THE PARALLEL REGION
Identify the code fragment that can be executed concurrently

WORK SHARING
Map compuational load (loop iterations) to threads,

Key App-oriented concepts

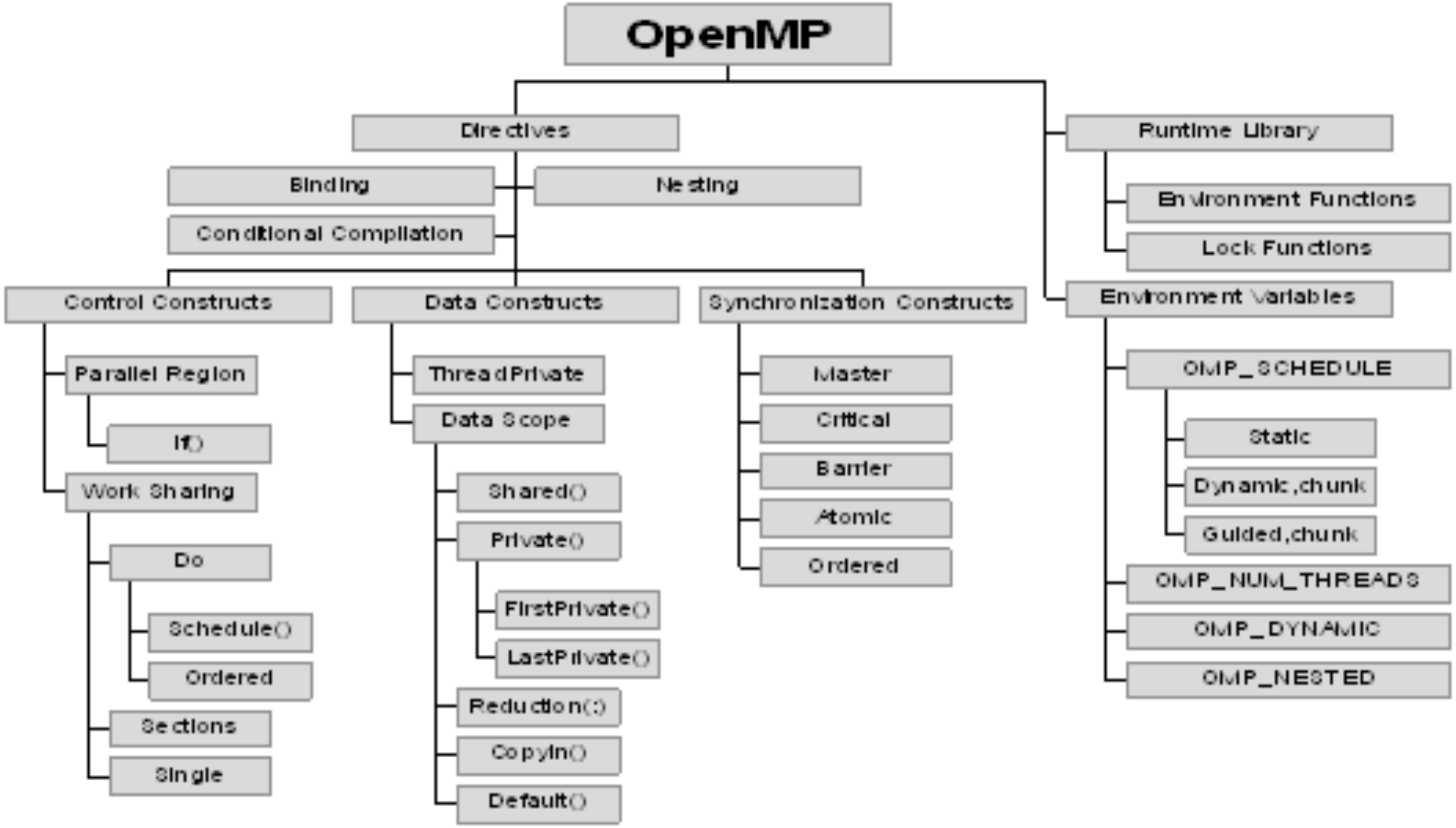
- **Parallel region**
  - Code fragment executed concurrently on sevral processors
- **Work-sharing**
  - Strategy to map computations to processors (e.g., ***block***, ***cyclic***).
- **Privatization**
  - Identify thread-local temporary values.
- **Synchronization**
  - Synch instructions between the threads to preserve the semantics of the program and avoid "***race conditions***" (p.ej., ***critical***, ***barrier***).
- **Reductions**
  - Identify computations that can be parallelized in three stages: distribution+computation+reconstruction.
  - Significant reduction in synch instructions, which raises performance.

MPI



MPI Function	Figure	Section	Illustrative Programs
MPI_INIT	8.1	8.2	8.1, 8.2, 8.3, etc.
MPI_FINALIZE	8.1	8.2	8.1, 8.2, 8.3, etc.
MPI_SEND	8.1	8.2	8.1, 8.2, 8.3, etc.
MPI_RECV	8.1	8.2	8.1, 8.2, 8.3, etc.
MPI_COMM_SIZE	8.1	8.2	8.1, 8.2, 8.3, etc.
MPI_COMM_RANK	8.1	8.2	8.1, 8.2, 8.3, etc.
MPI_IPROBE	8.6	8.4	8.6
MPI_GET_COUNT	8.6	8.4	8.6
MPI_PROBE	8.6	8.4	
MPI_BARRIER	8.2	8.3.1	
MPI_BCAST	8.2	8.3.2	8.5
MPI_GATHER	8.2	8.3.2	8.5
MPI_SCATTER	8.2	8.3.2	8.5
MPI_REDUCE	8.2	8.3.3	
MPI_ALLREDUCE	8.2	8.3.3	8.5
MPI_TYPE_CONTIGUOUS	8.12	8.6.1	8.8
MPI_TYPE_VECTOR	8.12	8.6.1	8.8
MPI_TYPE_INDEXED	8.12	8.6.1	
MPI_TYPE_COMMIT	8.12	8.6.1	8.8
MPI_TYPE_FREE	8.12	8.6.1	8.8
MPI_COMM_DUP	8.7	8.5.1	
MPI_COMM_SPLIT	8.7	8.5.2	8.7, 8.9
MPI_COMM_FREE	8.7	8.5.1	8.7
MPI_INTERCOMM_CREATE	8.7	8.5.3	8.7, 8.9

OpenMP



- OpenMP
  - Directives
    - Binding
    - Nesting
    - Conditional Compilation
    - Control Constructs
      - Parallel Region
        - If()
      - Work Sharing
        - Do
          - Schedule()
          - Ordered
        - Sections
        - Single
    - Data Constructs
      - ThreadPrivate
      - Data Scope
        - Shared()
        - Private()
          - FirstPrivate()
          - LastPrivate()
        - Reduction(:)
        - Copyin()
        - Default()
    - Synchronization Constructs
      - Master
      - Critical
      - Barrier
      - Atomic
      - Ordered
  - Runtime Library
    - Environment Functions
    - Lock Functions
  - Environment Variables
    - OMP_SCHEDULE
      - Static
      - Dynamic,chunk
      - Guided,chunk
    - OMP_NUM_THREADS
    - OMP_DYNAMIC
    - OMP_NESTED

OpenACC

CLAUSES	DIRECTIVES							
	parallel	kernels	data	loop	declare	update	wait	cache
if	x	x	x			x		
async	x	x	x			x		
num_gangs	x							
num_workers	x							
private	x							
firstprivate	x							
reduction	x			x				
create								
create/present copy/pcopy copyin/pcopyin copyout/pcopyout	x	x	x		x			
collapse				x				
gang/worker				x				
seq				x				
independent				x				
host/device						x		

Are there Hw-independent key concepts?

Are there frequently used design patterns for HPC Apps?

Assignment

- Modify the value of a set of memory locations overwriting the previous values:

  A = B

  The previous value is lost, it is not used to update the value saved in the memory location.

- Types of assignments:
  - Scalar assignment: s=1
  - Regular assignment: A(h)=1
  - Irregular assignment: A(f(h))=1

Scalar Assignment

- Code variant that uses temporary variables to store partial results.

```
w = 1/N
/* Cálculo del sumatorio */
sum = 0.0
#pragma omp parallel for \
                    reduction(+:sum) \
                    private(x)
for( i=0; i<N; i++ ) {
     x = (i - 0.5) / N
     sum = sum + ( 4.0 / ( 1+ x^2 ) )
}
/* Cálculo de PI */
pi = sum * w
```

Code: Computation of ∏

Regular Assignment

```
subroutine amuxe (y,a,x,nrows,ncols)
real*8  y(*), a(MaxNrows,*), x(*)
integer nrows, ncols(*)
integer i, j

#pragma omp parallel for shared(y)
do i = 1,nrows
    y(i) = 0.0
enddo
do i = 1,nrows
    do j= 1,ncols(i)
          y(i) = y(i) + a(i,j)*x(j)
    enddo
enddo
return
end
```

Code: Product of sparse matrix and dense vector.

Code:	Product of sparse matrix and dense vector.
Format:	Matrix without zeros at the end of the row.
Source:	Inspired in SparskitII, module MATVEC, routine amuxe.f (format ellpack-itpack)

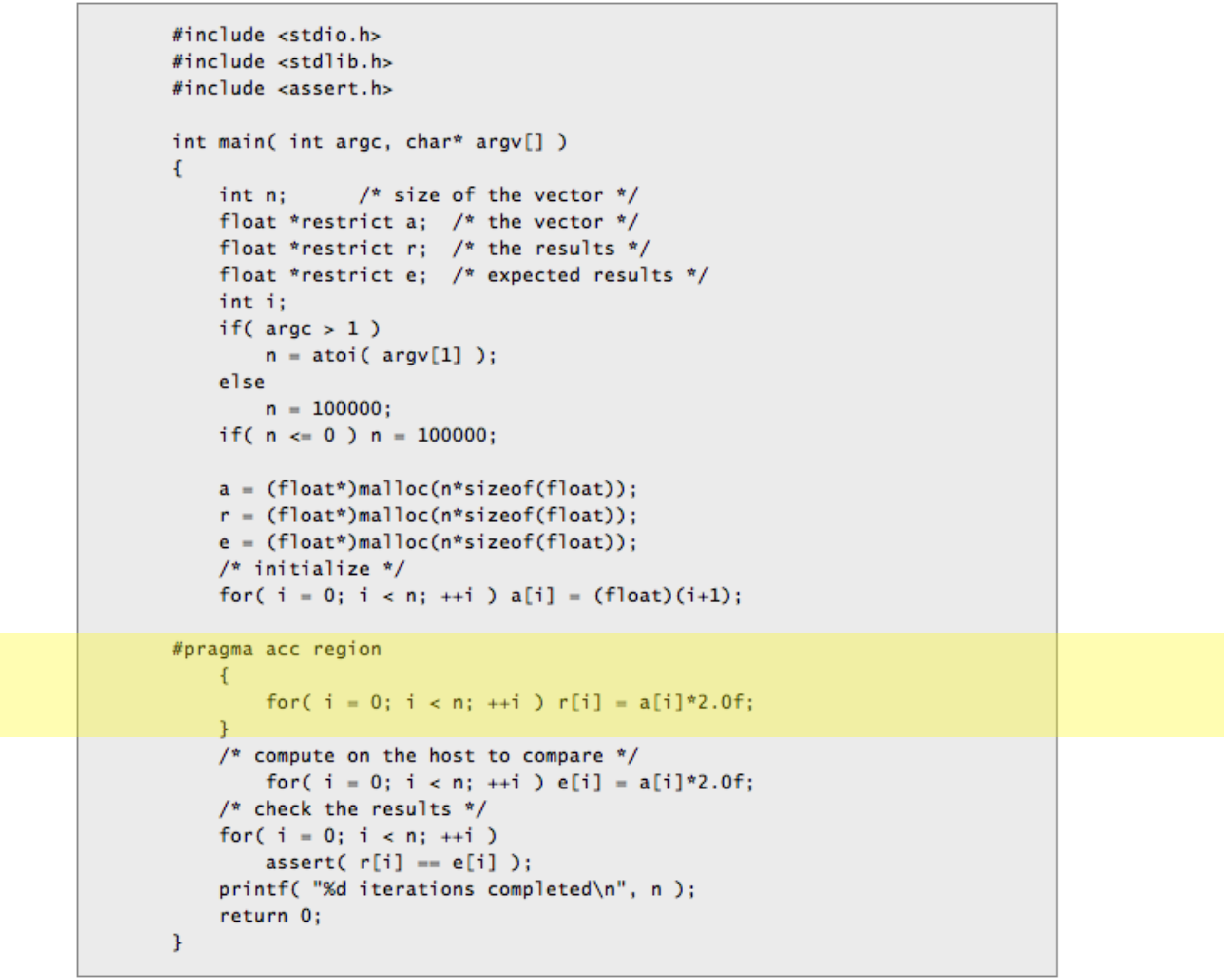
Regular Assignment

```
subroutine amux (n, x, y, a,ja,ia)
real*8  x(*), y(*), a(*)
integer n, ja(*), ia(*)
real*8 t
integer i, k

#pragma omp parallel for … \
          private(t)
do i = 1,n
    t = 0.0d0
    do k=ia(i), ia(i+1)-1
          t = t + a(k)*x(ja(k))
    enddo
    y(i) = t
enddo
return
end
```

```
subroutine amux (n, x, y, a,ja,ia)
real*8  x(*), y(*), a(*)
integer n, ja(*), ia(*)
integer i, k

#pragma omp parallel for … \
          shared(y)
do i = 1,n
    y(i) = 0.0d0
    do k=ia(i), ia(i+1)-1
          y(i) = y(i) + a(k)*x(ja(k))
    enddo
enddo
return
end
```

Code:	Product of sparse matrix by vector.
Format:	CRS sparse matrix.
Source:	SparskitII, module MATVEC, routine amux.f

Regular Assignment



```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

int main( int argc, char* argv[] )
{
    int n;      /* size of the vector */
    float *restrict a;  /* the vector */
    float *restrict r;  /* the results */
    float *restrict e;  /* expected results */
    int i;
    if( argc > 1 )
        n = atoi( argv[1] );
    else
        n = 100000;
    if( n <= 0 ) n = 100000;

    a = (float*)malloc(n*sizeof(float));
    r = (float*)malloc(n*sizeof(float));
    e = (float*)malloc(n*sizeof(float));
    /* initialize */
    for( i = 0; i < n; ++i ) a[i] = (float)(i+1);

#pragma acc region
    {
        for( i = 0; i < n; ++i ) r[i] = a[i]*2.0f;
    }
    /* compute on the host to compare */
        for( i = 0; i < n; ++i ) e[i] = a[i]*2.0f;
    /* check the results */
    for( i = 0; i < n; ++i )
        assert( r[i] == e[i] );
    printf( "%d iterations completed\n", n );
    return 0;
}
```

Reduction

- Update the value of a set of memory locations as a function of the previous value stored in each memory location:

  $$\mathbf{A = A \oplus B}$$

  The operator $\oplus$ is associative and commutative (e.g., sum, product, max/min).

- Types of reductions:
  - Scalar reduction: s=s+1
  - Regular reduction: A(h)=A(h)+1
  - Irregular reduction: A(f(h))=A(f(h))+1

Scalar Reduction

```
/* Cálculo del sumatorio */
sum = 0.0
#pragma omp parallel shared (N) private(i)\
                    reduction(+:sum)
{
#pragma for schedule(static)
for( i=0; i<N; i++ ) {
    sum = sum + ( 4.0 / ( 1+( (i - 0.5)/N )^2 ) )
}
}

/* Cálculo de PI */
pi = sum / N
```

Code: Computation of $\prod$

```
minlen = ia(2)-ia(1)
irow = 1
#pragma omp parallel for \
        reduction(minpos:minlen,irow) \
        private(len)
do i = 2,nrow
   len = ia(i+1)-ia(i)
   if (len.lt.minlen) then
         minlen = len
         irow = i
   endif
enddo
```

Code:	Computation of the value and the position of the minimum within an array
Format:	CRS sparse matrix
Source:	Inspired in SparskitII, module UNARY, routine blkfnd.f

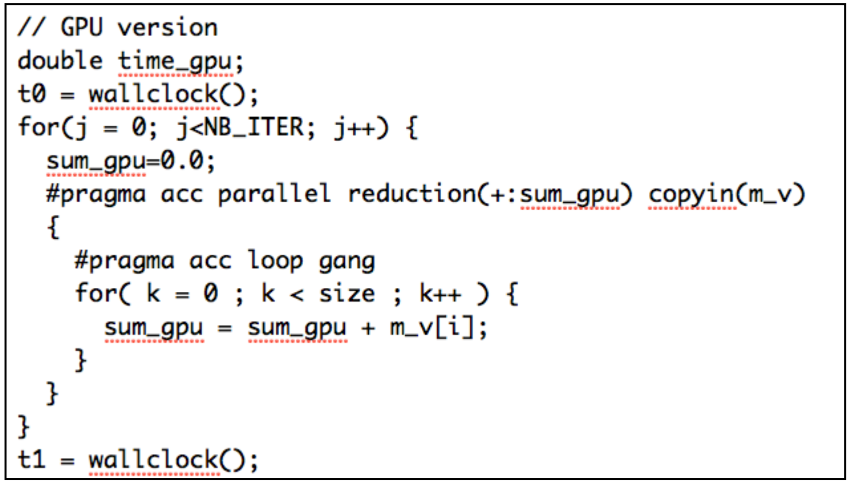
Scalar Reduction

```
/* Cálculo del contador */
cont=0
#pragma omp parallel private (x,y) \
                reduction(+:cont)
{
  #pragma omp for
  for( i=0; i<N; i++ ) {
    x=aleatorio()
    y=aleatorio()
    if( $x^2+y^2 \le 1$ ) {
      cont++
    }
  }
}
/* Cálculo de PI */
pi=4*cont/N
```

```
/* Cálculo del contador */
cont=0
#pragma omp parallel private (x,y) \
                reduction(+:cont)
{
  #pragma omp for schedule(static)
  for( i=0; i<N; i++ ) {
    x=aleatorio()
    y=aleatorio()
    if( $x^2+y^2 \le 1$ ) {
      cont++
    }
  }
}
/* Cálculo de PI */
pi=4*cont/N
```

```
/* Cálculo del contador */
cont=0
#pragma omp parallel private (x,y) \
                reduction(+:cont)
{
  #pragma omp for schedule(static,1)
  for( i=0; i<N; i++ ) {
    x=aleatorio()
    y=aleatorio()
    if( $x^2+y^2 \le 1$ ) {
      cont++
    }
  }
}
/* Cálculo de PI */
pi=4*cont/N
```

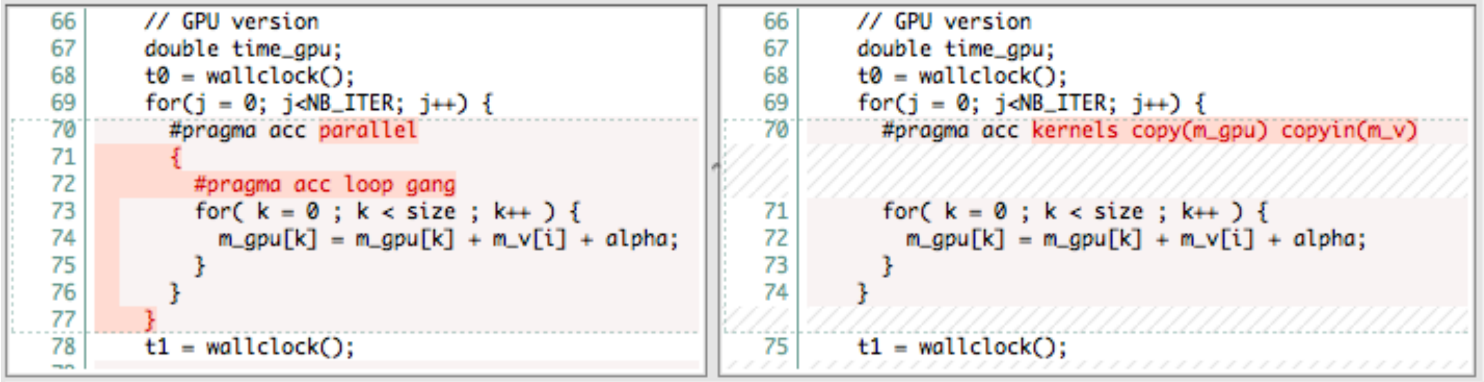
Scalar Reduction



```
// GPU version
double time_gpu;
t0 = wallclock();
for(j = 0; j<NB_ITER; j++) {
  sum_gpu=0.0;
  #pragma acc parallel reduction(+:sum_gpu) copyin(m_v)
  {
    #pragma acc loop gang
    for( k = 0 ; k < size ; k++ ) {
      sum_gpu = sum_gpu + m_v[i];
    }
  }
}
t1 = wallclock();
```

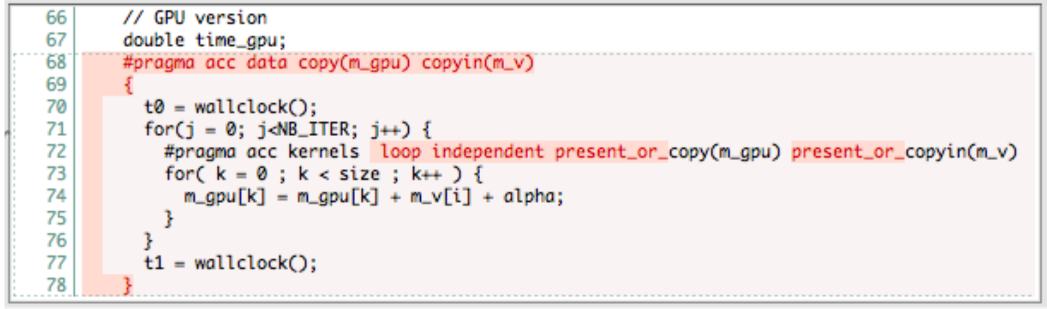
Code: Computation of the sum of the entries of an array.

Regular Reduction



```
    // GPU version
    double time_gpu;
    t0 = wallclock();
    for(j = 0; j<NB_ITER; j++) {
      #pragma acc parallel
      {
        #pragma acc loop gang
        for( k = 0 ; k < size ; k++ ) {
          m_gpu[k] = m_gpu[k] + m_v[i] + alpha;
        }
      }
    }
    t1 = wallclock();
```

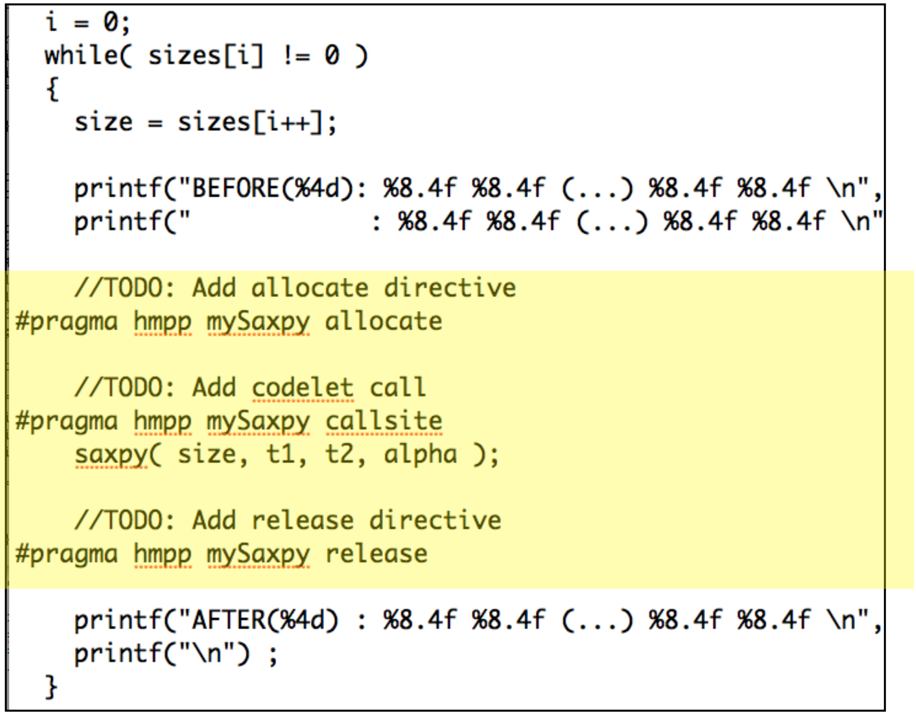
```
    // GPU version
    double time_gpu;
    t0 = wallclock();
    for(j = 0; j<NB_ITER; j++) {
      #pragma acc kernels copy(m_gpu) copyin(m_v)


      for( k = 0 ; k < size ; k++ ) {
        m_gpu[k] = m_gpu[k] + m_v[i] + alpha;
      }
    }

    t1 = wallclock();
```



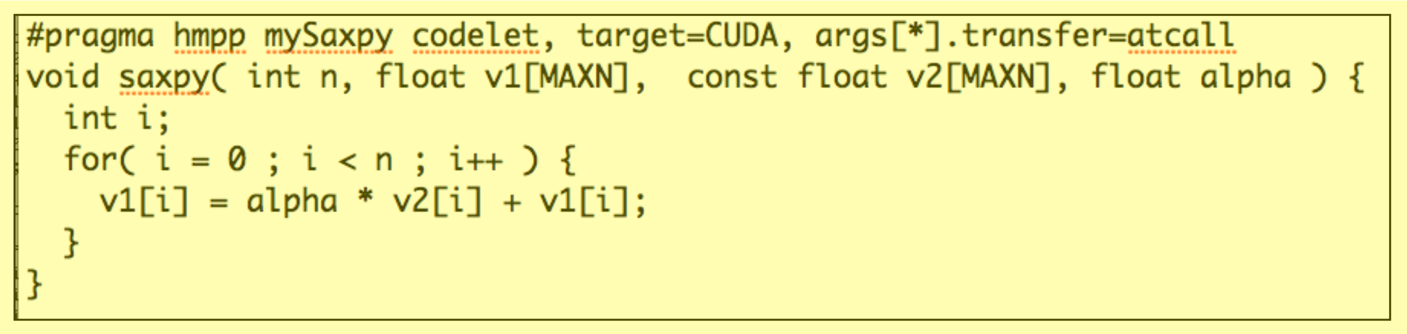
```
    // GPU version
    double time_gpu;
    #pragma acc data copy(m_gpu) copyin(m_v)
    {
      t0 = wallclock();
      for(j = 0; j<NB_ITER; j++) {
        #pragma acc kernels  loop independent present_or_copy(m_gpu) present_or_copyin(m_v)
        for( k = 0 ; k < size ; k++ ) {
          m_gpu[k] = m_gpu[k] + m_v[i] + alpha;
        }
      }
      t1 = wallclock();
    }
```

Code: Vector addition and accumulation

Regular Reduction



```
  i = 0;
  while( sizes[i] != 0 )
  {
    size = sizes[i++];

    printf("BEFORE(%4d): %8.4f %8.4f (...) %8.4f %8.4f \n",
    printf("             : %8.4f %8.4f (...) %8.4f %8.4f \n"

    //TODO: Add allocate directive
#pragma hmpp mySaxpy allocate

    //TODO: Add codelet call
#pragma hmpp mySaxpy callsite
    saxpy( size, t1, t2, alpha );

    //TODO: Add release directive
#pragma hmpp mySaxpy release

    printf("AFTER(%4d) : %8.4f %8.4f (...) %8.4f %8.4f \n",
    printf("\n") ;
  }
```



```
#pragma hmpp mySaxpy codelet, target=CUDA, args[*].transfer=atcall
void saxpy( int n, float v1[MAXN],  const float v2[MAXN], float alpha ) {
  int i;
  for( i = 0 ; i < n ; i++ ) {
    v1[i] = alpha * v2[i] + v1[i];
  }
}
```

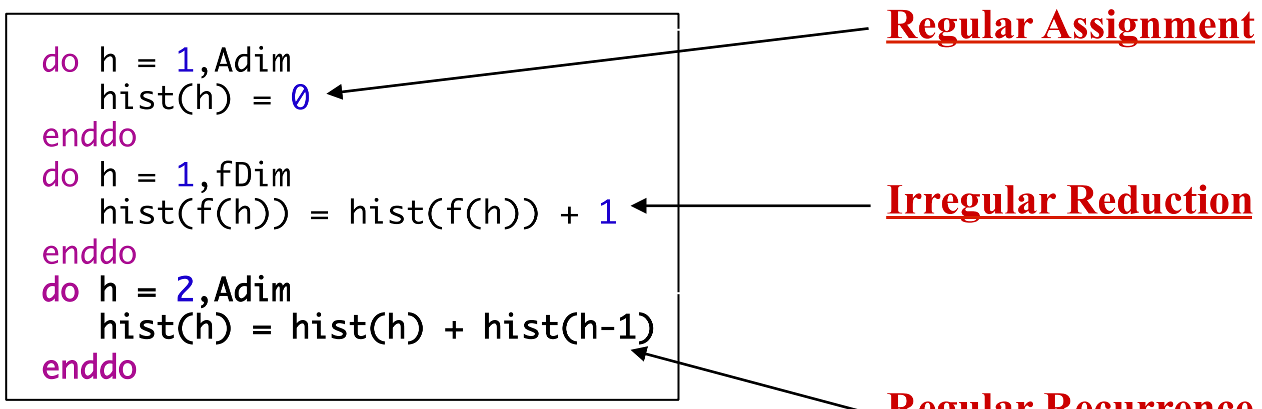
Recurrences

- Update the value of a set of memory locations as a function of the previous values stored in several the memory locations:

$$\mathbf{A = A \oplus \ldots \oplus A \oplus B}$$

- Types of recurrences:
  - Regular recurrences: $A(h)=A(h-1)+1$
  - Irregular recurrences: $A(f(h))=A(g(h))+1$

Recurrences



```
do h = 1,Adim
   hist(h) = 0
enddo
do h = 1,fDim
   hist(f(h)) = hist(f(h)) + 1
enddo
do h = 2,Adim
   hist(h) = hist(h) + hist(h-1)
enddo
```

Regular Assignment

Irregular Reduction

Regular Recurrence

Code: Computation of the accumulative histogram of an irregular access pattern.

Source: Inspector with load-balancing used in the execution of parallel irregular reductions.

Recurrences

```
subroutine lsol (n,x,y,al,jal,ial)
integer n, jal(*),ial(n+1)
real*8  x(n), y(n), al(*)
integer k, j
real*8  t

x(1) = y(1)
do k = 2, n
   t = y(k)
   do j = ial(k), ial(k+1)-1
      t = t-al(j)*x(jal(j))
   enddo
   x(k) = t
enddo
return
end
```

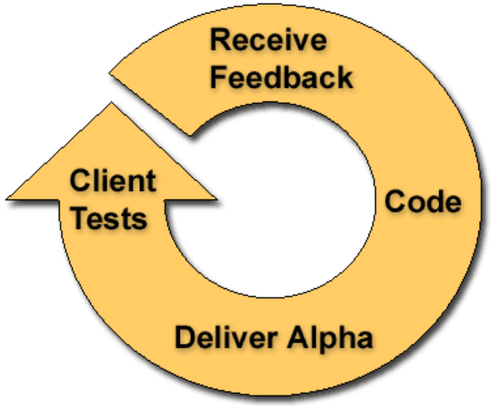
Code: Solver of a system of equations;
standard forward-elimination method.
Format: Lower triangular unit matix; CRS format.
Source: SparskitII, module MATVEC, routine lsol.f

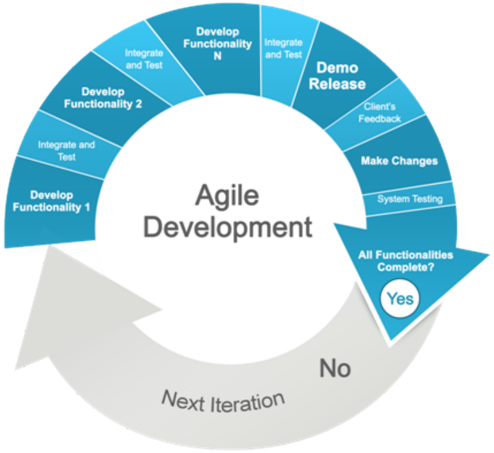
Are there frequently used design patterns for HPC Apps?

Can you propose a development methodology for GPU programming?

Agile Development

- □ Reduce uncertainty as much as possible.
- □ Get feedback from the market to avoid leaps-of-faith.
- □ Run experiments following the cycle Build-Meassure-Learn.
- □ Methodology:
  - ■ Prototyping & Incremental
  - ■ Sprints
  - ■ Evertything is a deliverable
  - ■ E.g. SCRUM
- □ Related works:
  - ■ Business model canvas
  - ■ Lean startup model





Devel Methodology

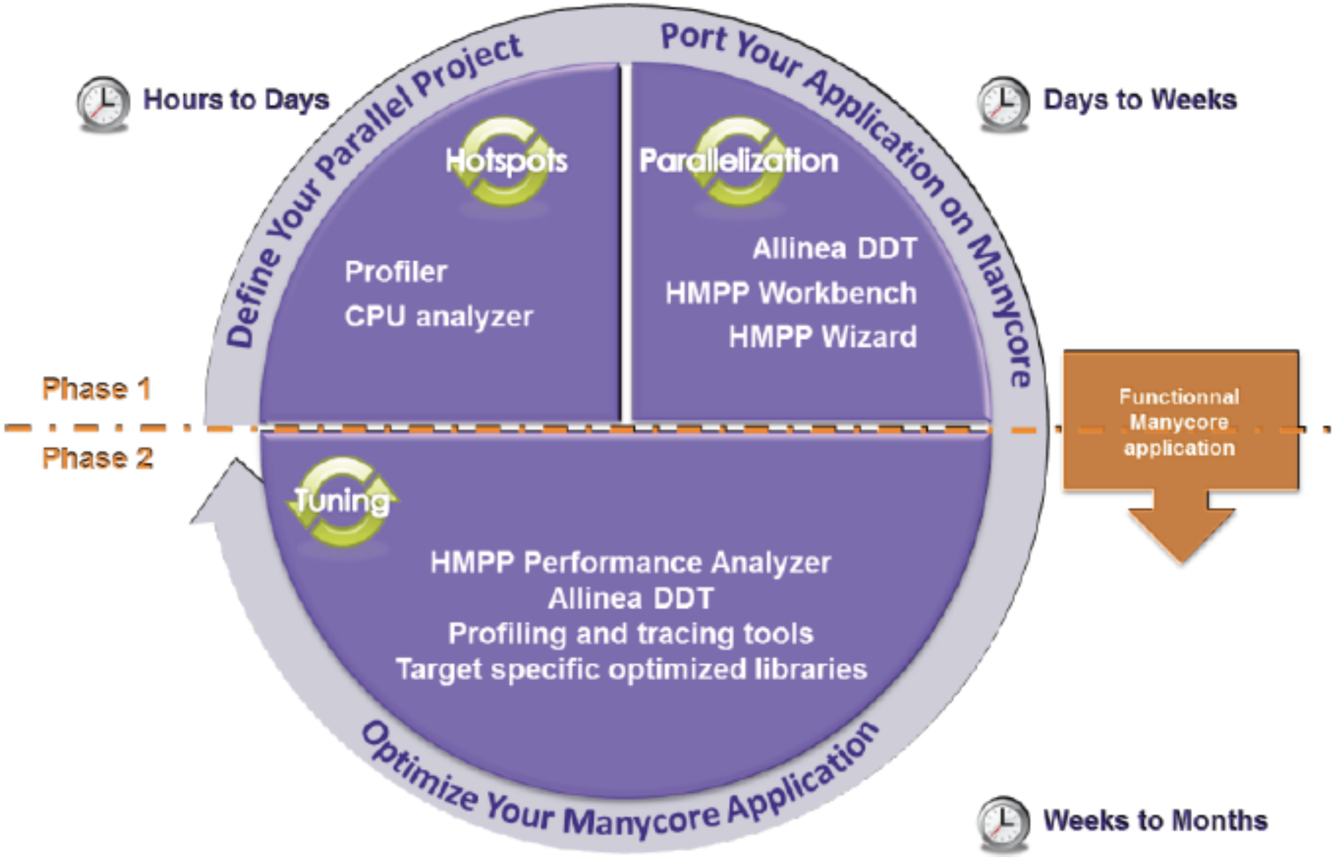

Hours to Days
Define Your Parallel Project
Hotspots
Profiler
CPU analyzer
Port Your Application on Manycore
Days to Weeks
Parallelization
Allinea DDT
HMPP Workbench
HMPP Wizard
Phase 1
Phase 2
Functionnal Manycore application
Tuning
HMPP Performance Analyzer
Allinea DDT
Profiling and tracing tools
Target specific optimized libraries
Optimize Your Manycore Application
Weeks to Months

FIGURE 8: METHODOLOGY OVERVIEW.

Courtesy: CAPS
HMPP CoC Europe
“Write once, deploy many-core”

Devel Methodology for GPUs

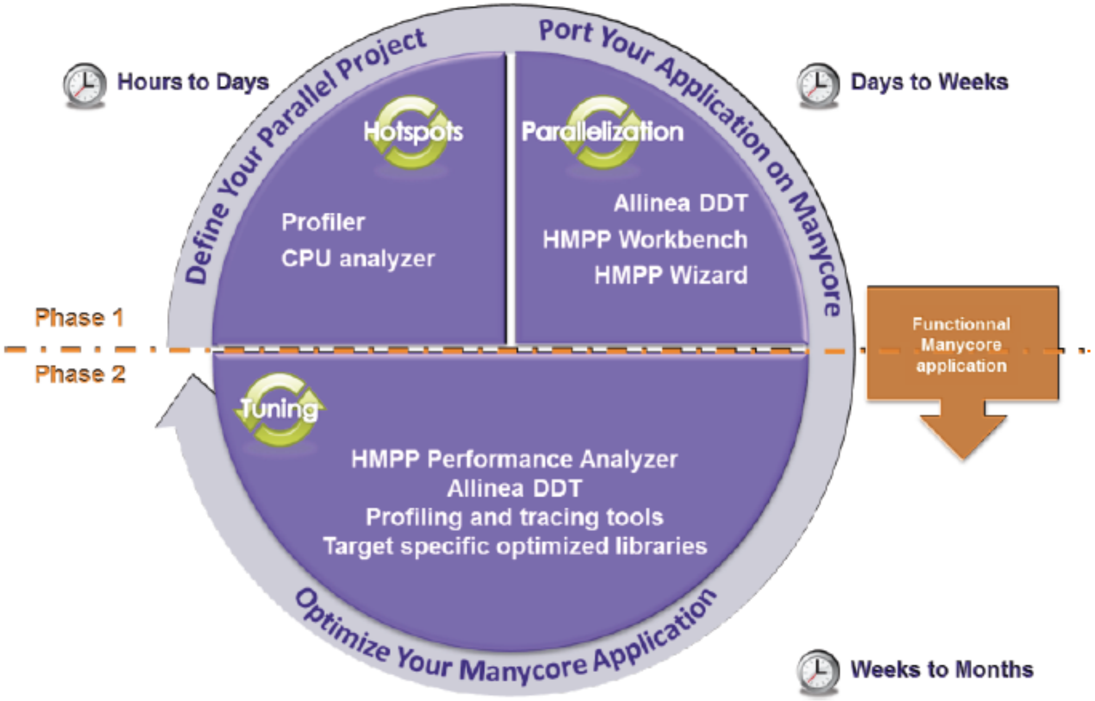
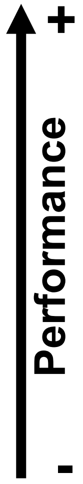

Hours to Days
Define Your Parallel Project
Port Your Application on Manycore
Days to Weeks
Hotspots
Profiler
CPU analyzer
Parallelization
Allinea DDT
HMPP Workbench
HMPP Wizard
Phase 1
Phase 2
Functionnal Manycore application
Tuning
HMPP Performance Analyzer
Allinea DDT
Profiling and tracing tools
Target specific optimized libraries
Optimize Your Manycore Application
Weeks to Months

FIGURE 8: METHODOLOGY OVERVIEW.

Courtesy: CAPS
HMPP CoC Europe
“Write once, deploy
many-core”

- **Extract parallelism**
- **Streamize source code**
- **Data transfers**
- **Race conditions**
- **Privatization**
- **Data dependences**

1. **Coalescence**
2. **GPU BlockSize**
3. **Register pressure**
4. **Shared memory**
5. **Bank conflicts**
6. **Texture memory**



Can you propose a development methodology for GPU programming?

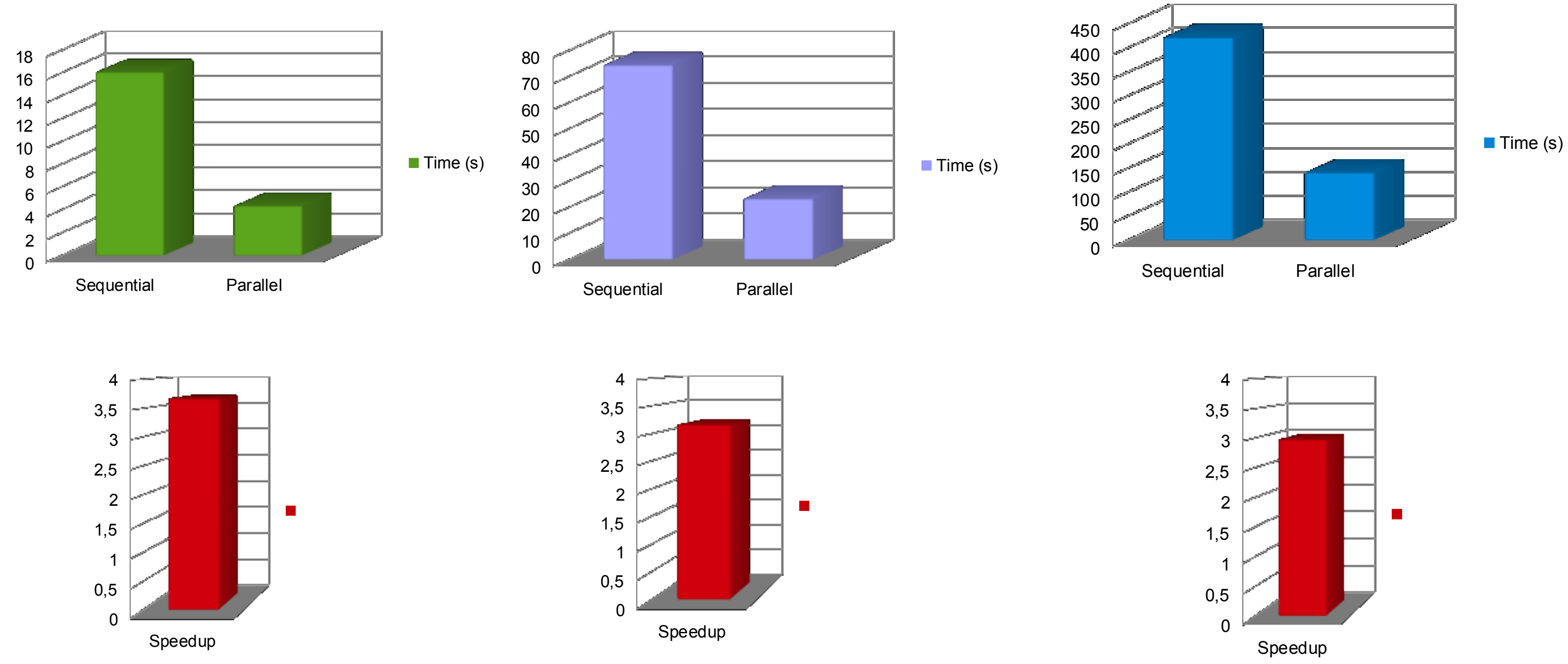
Can you meassure productivity?

Experiment

- Benchmarks:
  - Sobel: Sobel edge filter written in C
  - Matmul: Matrix-Matrix product written in C
  - Laplace2D: Laplace transformation written in C
- Team skills&expertise:
  - Master degree in Computer Science
  - Basic skills in parallel programming: PVM/MPI, OpenMP
- Results:
  - Performance (speedup)
  - Development time (days)

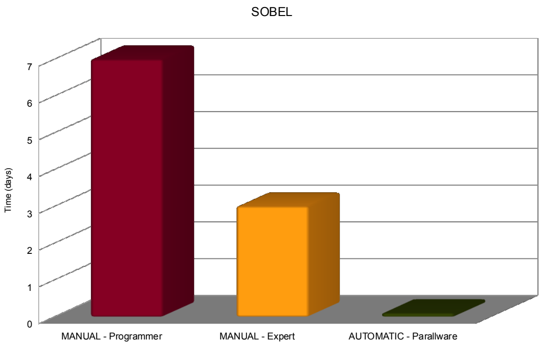
Performance (Intel i7 4xcore)

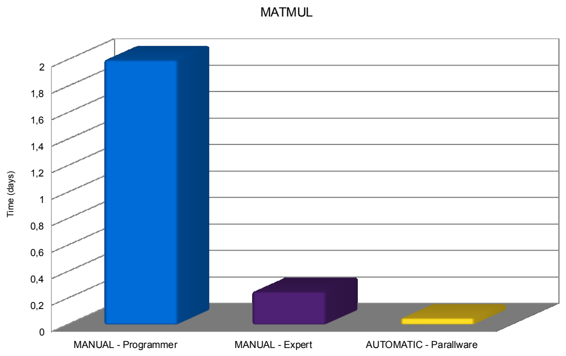
Sobel Matmul Laplace2D

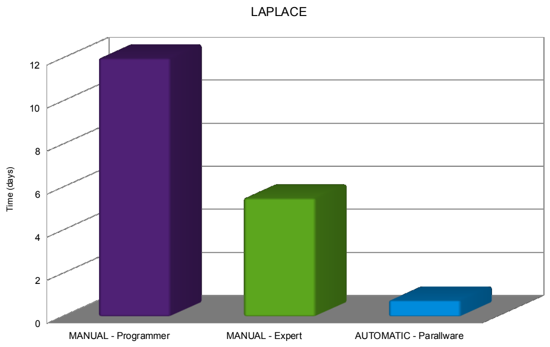


Development Time

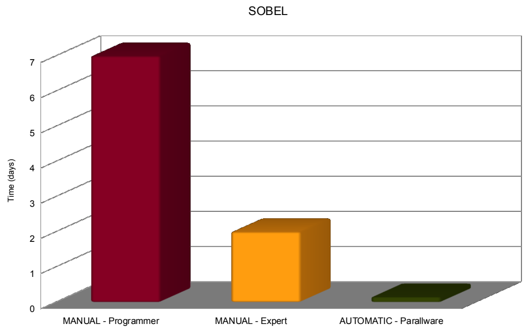
OpenMP

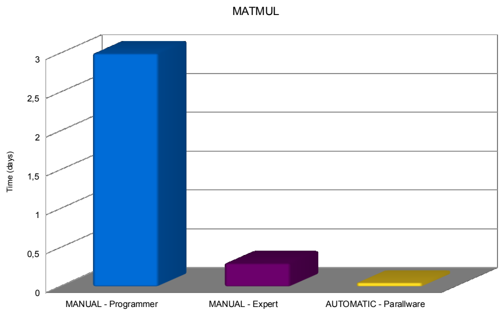


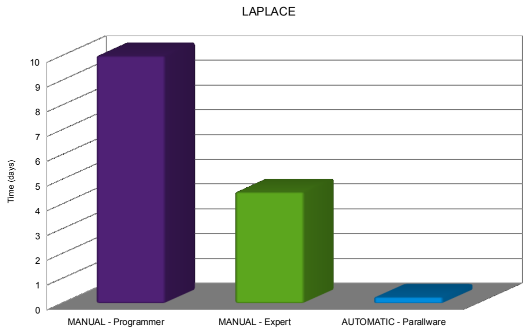




OpenACC







Can you meassure productivity?

Conclusions

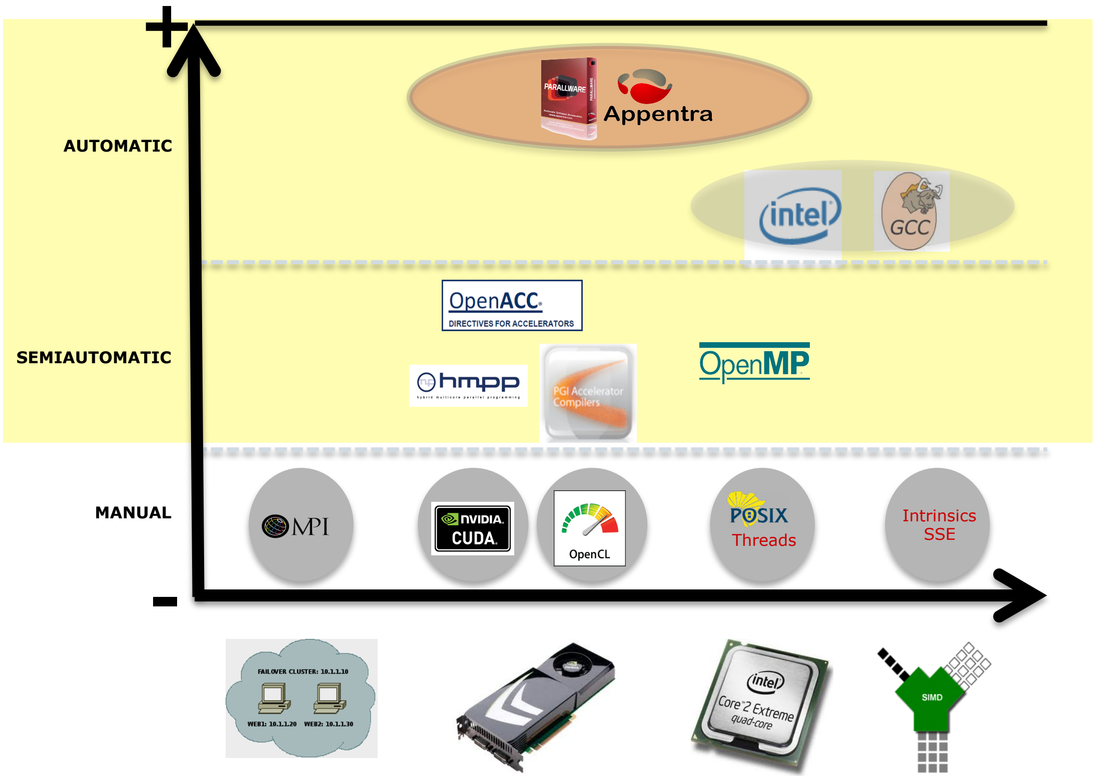
Increase Your Productivity

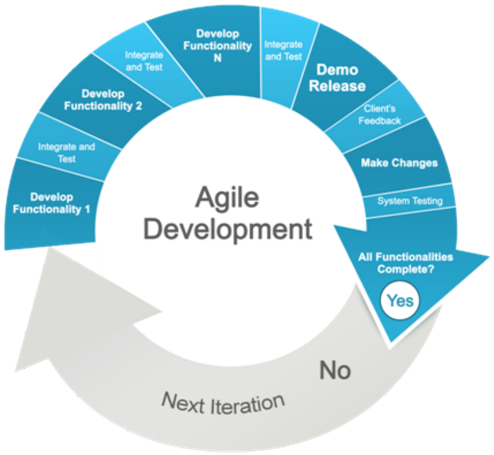
- Increasing programmer's productivity is a must. Whenever possible:
  - Use automatic or semiautomatic tools
  - Avoid manual tools that require deep knowledge of Hw and HPC techniques

- More sophisticated software tools are needed
  - Appentra's ParallWare is a step forward towards fully automation

Avoid leaps-of-faith

Engineer
Scientist
Researcher





Parallel Design Patterns

- Parallel design patterns are effective
- The complexity lays in identifying the parallel design pattern in real Apps
  - An algorithm may be coded in an unlimited number of different ways
  - Handle pointers, calls, complex control flows, dynamic memory, sparse computations, etc…

GPUs & Productivity

- OpenACC is a promising approach
  - Based on compiler directives
- Fast development of hybrid CPU/GPU Apps
  - For C and Fortran programming languages
- Beware of compiler-dependent behaviors
  - CAPS HMPP (first commercial version)
  - PGI ACC (by the end of this year)
- Use CUDA/OpenCL for more control and more performance

GPU Programming and Productivity in Software Development

MANUEL ARENAZ

UNIVERSIDADE DA CORUÑA
Associate Professor
manuel.arenaz@udc.es

Appentra
CTO & Co-founder
manuel.arenaz@appentra.com

V Jornadas de Supercomputación y Avances en Tecnología COMPUTAEX

Cáceres, November 19-20, 2012

http://www.cenits.es/noticias/241012-fundacion-computaex-organiza-v-jornadas-supercomputacion-avances-tecnologia