Προγραμματισμός Καρτας γραφικών με OpenMP OpenACC

ΠΑΡΆΛΛΗΛΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΌΣ ΣΕ ΠΡΟΒΛΉΜΑΤΑ ΜΗΧΑΝΙΚΉΣ ΜΆΘΗΣΗΣ

ΠΡΟΓΡΑΜΜΑΤΙΣΜΌΣ ΣΥΣΤΗΜΆΤΩΝ ΥΨΗΛΉΣ ΕΠΊΔΟΣΗΣ

ΠΟΤΕ ΧΡΗΣΙΜΟΠΟΙΩ ΤΗΝ ΚΑΡΤΑ ΓΡΑΦΙΚΩΝ;

1. Μικρά προγράμματα
2. Λιγες μεταφορές δεδομένων προς/από κάρτα γραφικλών
3. Λίγα if
4. Προτιμούνται ακέραιες/float μεταβλητές
5. Πράξεις με λιγότερη αριθμητική ακρίβεια σε float
6. Μικρότερη κατανάλωση ενέργειας για τον ίδιο αριθμό υπολογισμών
7. Χαμηλότερο κόστος υλικού
8. Χαμηλότερες ταχύτητες επεξεργαστών

9 Υψηλότερες ταχύτητες προσπέλασης μνήμης

10 Απλούστερη γλώσσα μηχανής (RISC)

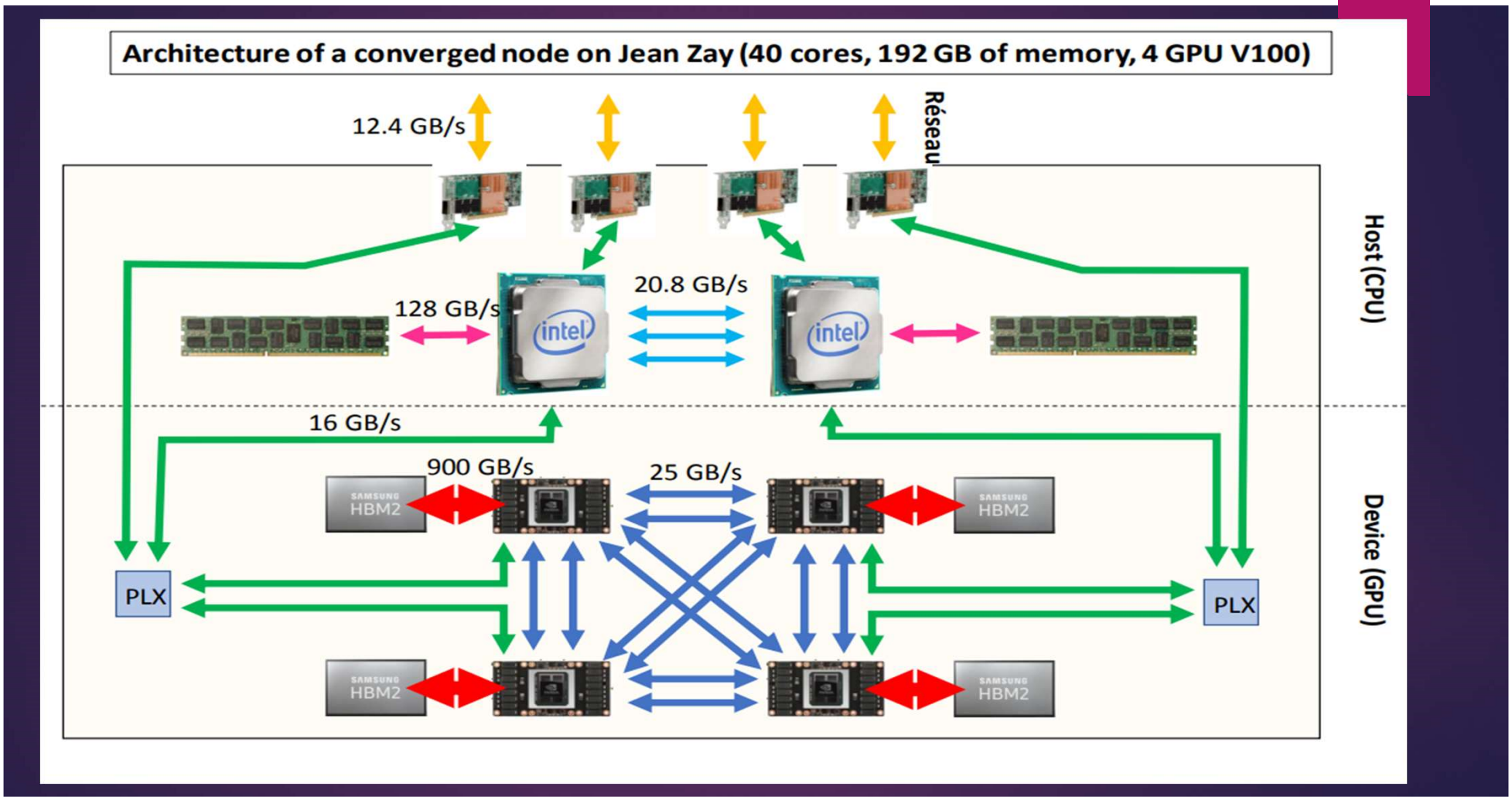
OPENMP VS OPENACC

OpenMP target

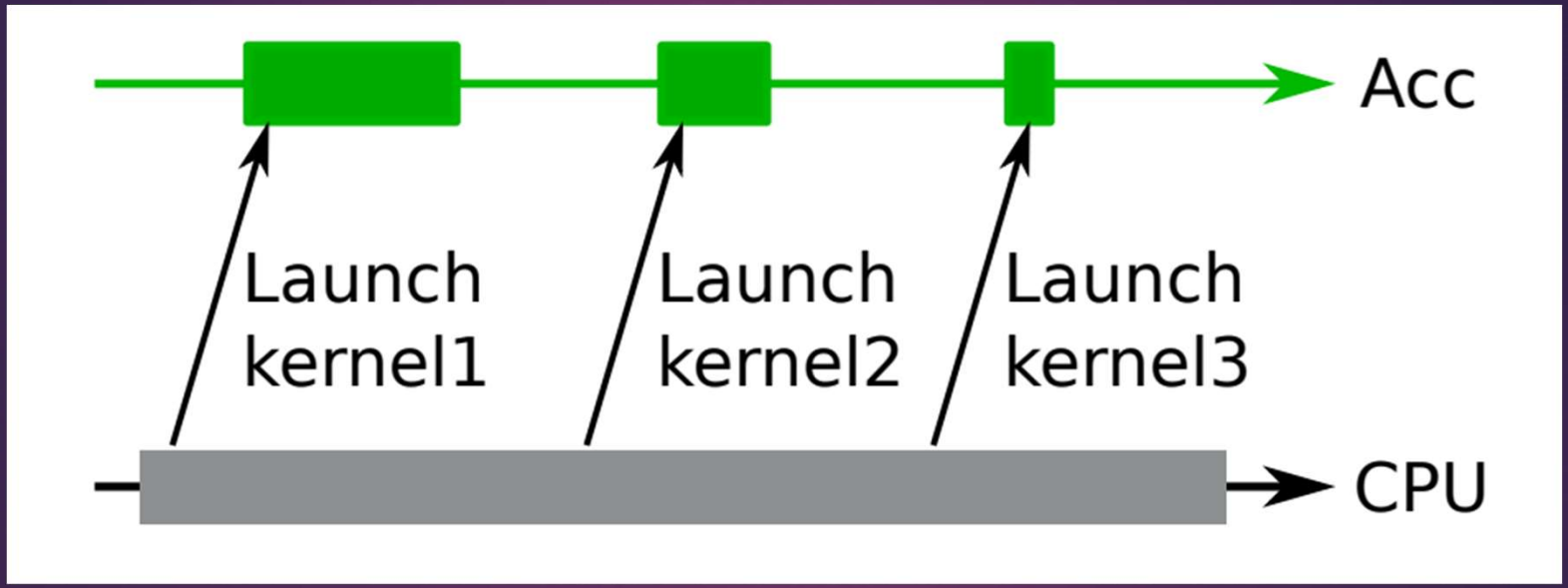
- `http://www.openmp.org/`
- First standard OpenMP 4.5 (11/2015)
- Latest standard OpenMP 5.0 (11/2018)
- Main compilers :
  - Cray (for Cray hardware)
  - GCC (since 7)
  - CLANG
  - IBM XL
  - PGI support announced

OpenACC

- `http://www.openacc.org/`
- Cray, NVidia, PGI, CAPS
- First standard 1.0 (11/2011)
- Latest standard 3.0 (11/2019)
- Main compilers :
  - PGI
  - Cray (for Cray hardware)
  - GCC (since 5.7)


Architecture of a converged node on Jean Zay (40 cores, 192 GB of memory, 4 GPU V100)
12.4 GB/s
Réseau
Host (CPU)
128 GB/s
20.8 GB/s
intel
intel
16 GB/s
900 GB/s
25 GB/s
SAMSUNG HBM2
SAMSUNG HBM2
SAMSUNG HBM2
SAMSUNG HBM2
PLX
PLX
Device (GPU)

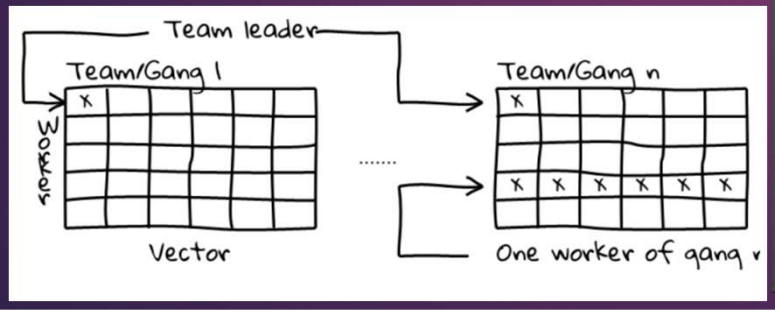
ΕΚΤΕΛΕΣΗ ΥΠΟΛΟΓΙΣΜΩΝ ΣΤΗΝ ΚΑΡΤΑ ΓΡΑΦΙΚΩΝ

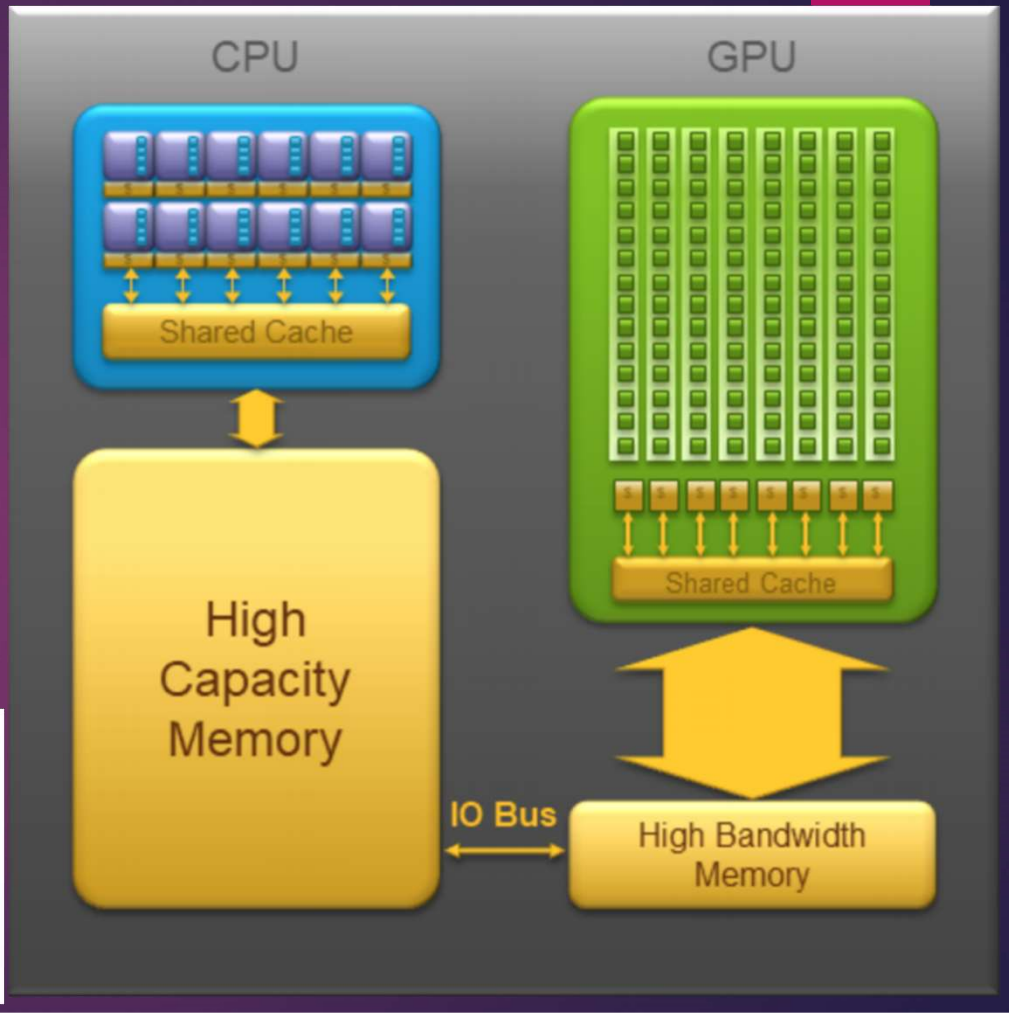


OPENMP VS OPENACC

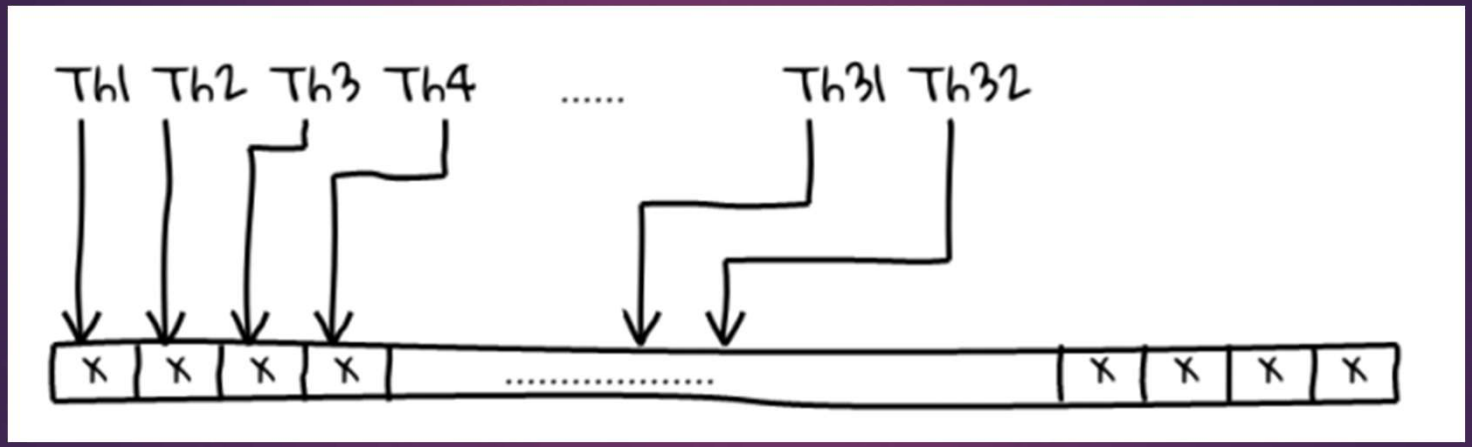
CPU-Υπολογιστική Ιεραρχία
Core
Threads
SIMD

GPGPU-Υπολογιστική Ιεραρχία
Gang
Workers
Vector

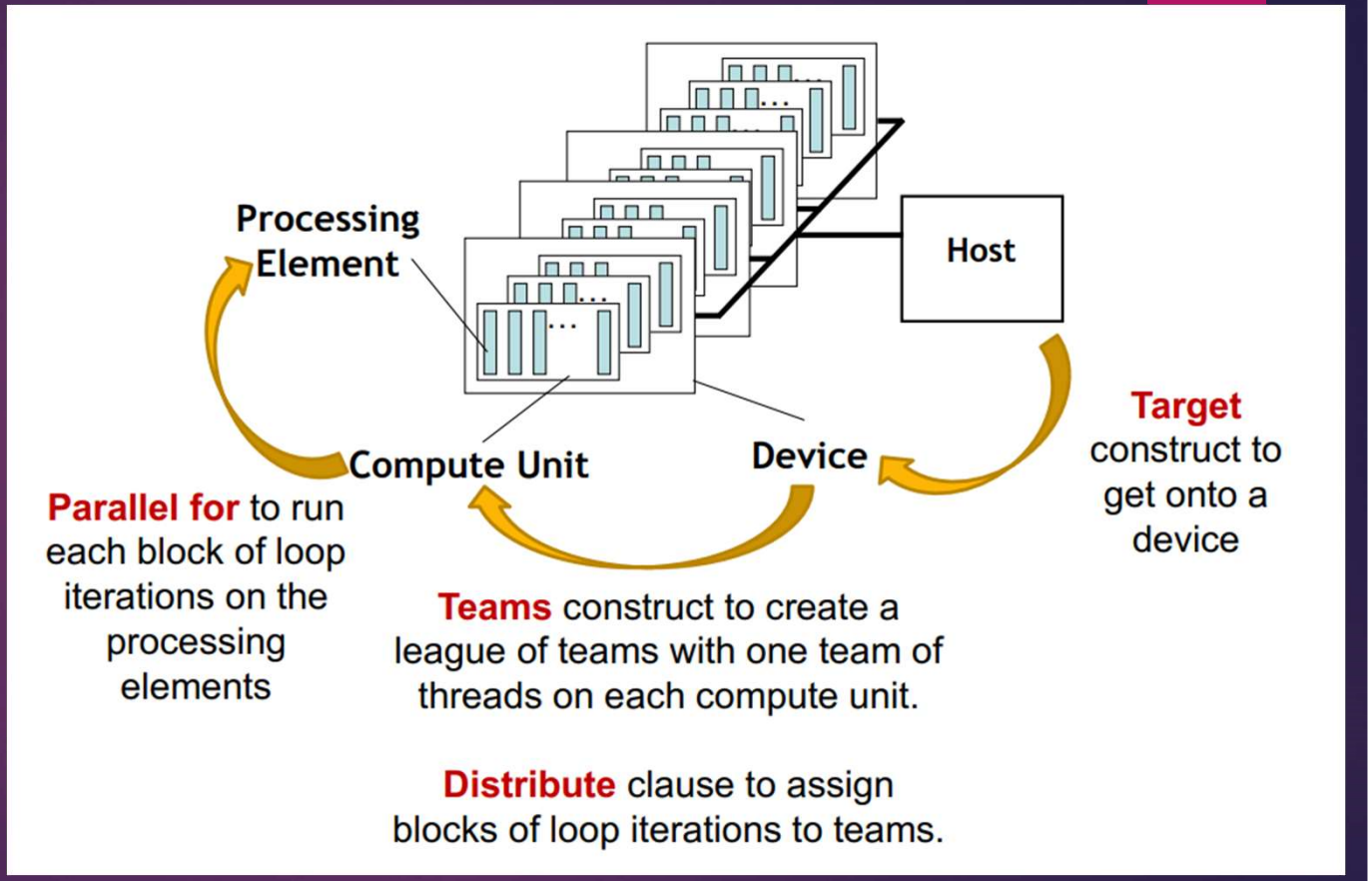




OPENMP KAI OPENACC: VECTOR PROCESSING



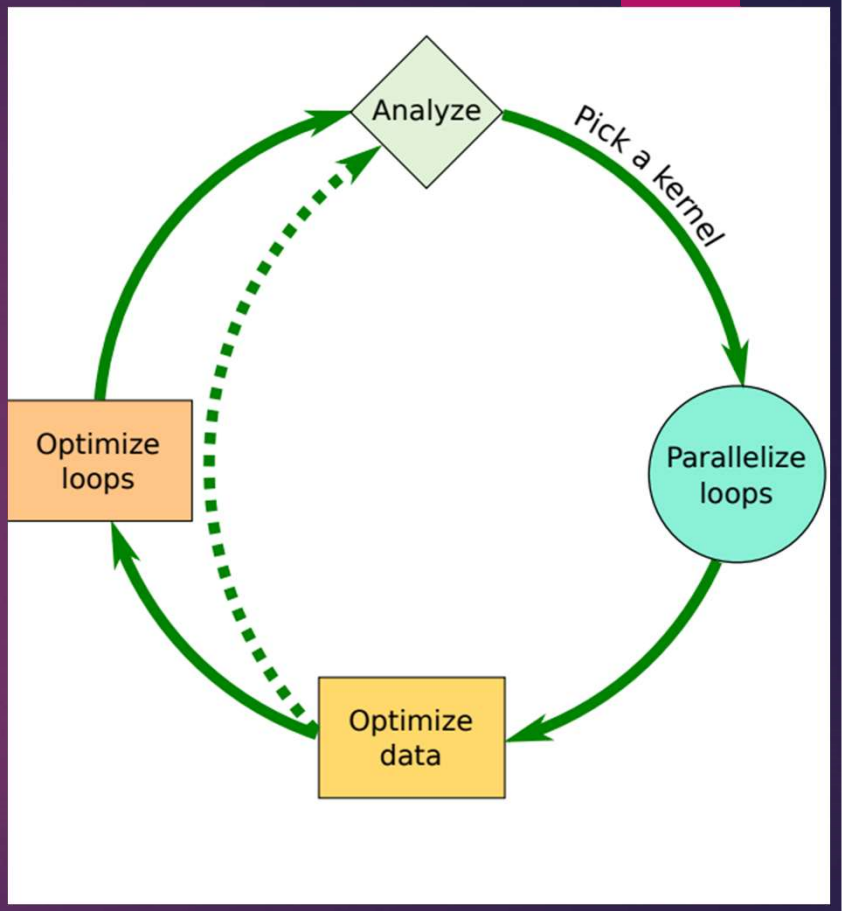
OPENMP - GPGPU



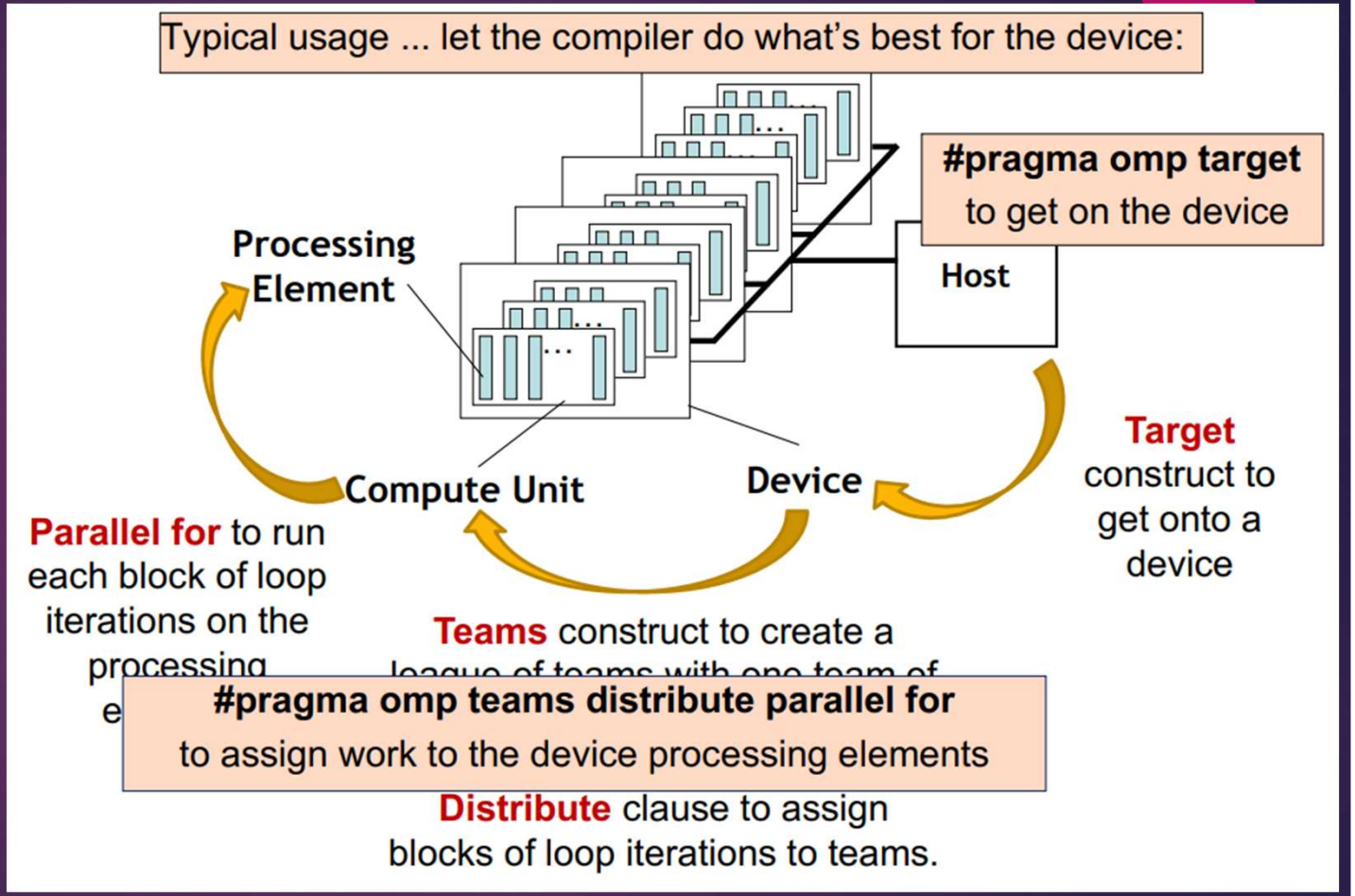
OPENMP ΚΑΙ OPENACC

Καλή Στρατηγική

1. Φτιάχνω τον σειριακό κώδικα
2. Προσθετω τις οδηγίες της OpenMP, OpenACC
3. Ελέγχω την απόδοση, σωστά αποτελέσματα
4. Ελαχιστοποιώ τις μεταφορές δεδομένων από μνήμη CPU σε μνήμη GPGPU
5. Ελέγχω την απόδοση , σωστά αποτελέσματα
6. Επαναλαμβάνω από 2



OPENMP KAI GPGPU



OPENMP GPGPU ΚΑΙ ΔΕΙΚΤΕΣ (POINTERS)

```
#pragma omp target map(…)

int main(void) {
Int A[N] ;
int* B = (int*)malloc(sizeof(int)*N);
for( i = 0 ; i < N ; i++ )
      B[i] = i+1 ;
#pragma omp target
#pragma omp teams distribute parallel for
for(int i = 0; i < N; ++i) {
….   // Ο πίνακας Α η μεταβλητή i  και ο
δείκτης Β υπάρχουν αλλά δεν έχουν
μεταφερθεί τα δεδομένα του Β
}
```

```
#pragma omp target map(…)

int main(void) {
Int A[N] ;
int* B = (int*)malloc(sizeof(int)*N);
for( i = 0 ; i < N ; i++ )
      B[i] = i+1 ;
#pragma omp target map(B[0:N])
#pragma omp teams distribute parallel for
for(int i = 0; i < N; ++i) {
….   // Ο πίνακας Α η μεταβλητή i  και ο
δείκτης Β υπάρχουν αλλά δεν έχουν
μεταφερθεί τα δεδομένα του Β
}
```

OPENMP ΚΑΙ GPGPU

Οδηγίες

#pragma omp target
#pragma omp target data
#pragma omp target update
#pragma omp declare target
#pragma omp teams
#pragma omp distribute
#pragma omp parallel for

#pragma omp target data
#pragma omp target enter data
#pragma omp target exit data
#pragma omp target simd
Σημαντικό
#pragma omp target nowait

```
#pragma omp target data map(to:c[0:N],
b[0:N],s) map(from: a[0:N])
#pragma omp target teams distribute parallel for
for (j=0; j<N; j++)
      a[j] = b[j]+s*c[j];
```

```
#pragma omp target data map (to: A[0:N*N],B[0:N*N])
map (tofrom: C[0:N*N])
#pragma omp target
#pragma omp teams distribute parallel for collapse(3)
   for(int i=0;i<N;i++)
      for(int j=0;j<N;j++)
         for(int k=0;k<N;k++)
            C[i*N+j]+=A[i*N+k]*B[k*N+j];
```

OPENMP ΚΑΙ OPENACC: REDUCTION

Μεταβλητές

Char
Unsigned char
Int
Unsigned int
Float
Double
_Complex

Operation	Effect	Language(s)
+	Sum	Fortran/C(++)
*	Product	Fortran/C(++)
max	Maximum	Fortran/C(++)
min	Maximum	Fortran/C(++)
&	Bitwise and	C(++)
|	Bitwise or	C(++)
&&	Logical and	C(++)
||	Logical or	C(++)

OPENMP ΚΑΙ GPGPU

#pragma omp target map(…)

map(to:….) Δεδομενα μεταφέρονται από την κύρια μνήμη (host memory) στην μνήμη της GPU
map(from:….) Στο τέλος των υπολογισμών της GPU, δεδομένα μεταφέρονται στην κύρια μνήμη
map(tofrom:….) και τα δυο προηγούμενα
Map(alloc:…..) Οι μεταβλητες δημιουργούνται στην μνήμη της GPU μόνο
map(….) ισοδύναμο με map(tofrom:….)
Οι δείκτες στην μνήμη της GPU οριζονται σαν πίνακες
map(to:p[0:N]), με N αυθαίρετο διότι δεν έχει φυσικό νόημα

Επιτρέπονται πολλά map ταυτόχρονα
#pragma omp target map(to: A[0:N],b,C[0:14]) map(tofrom: D[0:N]) map(alloc: S[0:N],q[0:14])

ΠΕΡΙΤΤΕΣ ΜΕΤΑΦΟΡΕΣ ΔΕΔΟΜΕΝΩΝ

```
#pragma omp target map(tofrom:xnew[0:Ndim],xold[0:Ndim]) \
                   map(to:A[0:Ndim*Ndim], b[0:Ndim])
  #pragma omp teams distribute parallel for
  for (int i=0; i<Ndim; i++){
      xnew[i] = (TYPE) 0.0;
      for (int j=0; j<Ndim;j++){
          if(i!=j)
            xnew[i]+= A[i*Ndim + j]*xold[j];
      }
      xnew[i] = (b[i]-xnew[i])/A[i*Ndim+i];
  }
  conv = 0.0;
#pragma omp target map(to:xnew[0:Ndim],xold[0:Ndim]) \
                   map(tofrom:conv)
   #pragma omp teams distribute parallel for reduction(+:conv)
   for (int i=0; i<Ndim; i++){
       tmp  = xnew[i]-xold[i];
       conv += tmp*tmp;
   }
conv = sqrt((double)conv);
// Pointer swap…
}
```

Each iteration, copy $(3*Ndim+Ndim^2)*sizeof(TYPE)$ bytes to device

Each iteration, copy 2*Ndim*sizeof(TYPE) bytes from device

Each iteration, copy 2*Ndim*sizeof(TYPE) bytes to device

ΜΕΤΑΦΟΡΕΣ ΔΕΔΟΜΕΝΩΝ ΑΝΕΞΑΡΤΗΤΑ ΑΠΌ ΤΗΝ ΕΚΤΕΛΕΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

```
#pragma omp target enter data map(to: A[0:N],B[0:N],C[0:N])

#pragma omp target
{do lots of stuff with A, B and C}
{do something on the host}
#pragma omp target
{do lots of stuff with A, B, and C}

#pragma omp target exit data map(from: C[0:N])
```

ΕΝΗΜΕΡΩΣΗ ΔΕΔΟΜΕΝΩΝ ΑΝΕΞΑΡΤΗΤΑ ΑΠΌ ΤΗΝ ΕΚΤΕΛΕΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

```
#pragma omp target enter data map(to: A[0:N],B[0:N],C[0:N])

#pragma omp target
{do lots of stuff with A, B and C}

#pragma omp update from(A[0:N])

{do something on the host}

#pragma omp update to(A[0:N])

#pragma omp target
{do lots of stuff with A, B, and C}

#pragma omp target exit data map(from: C[0:N])
```

Copy A on the device to A on the host.

Copy A on the host to A on the device.

Branching

Conditional execution

```
// Only evaluate expression
// if condition is met
if (a > b)
{
  acc += (a - b*c);
}
```

Selection and masking

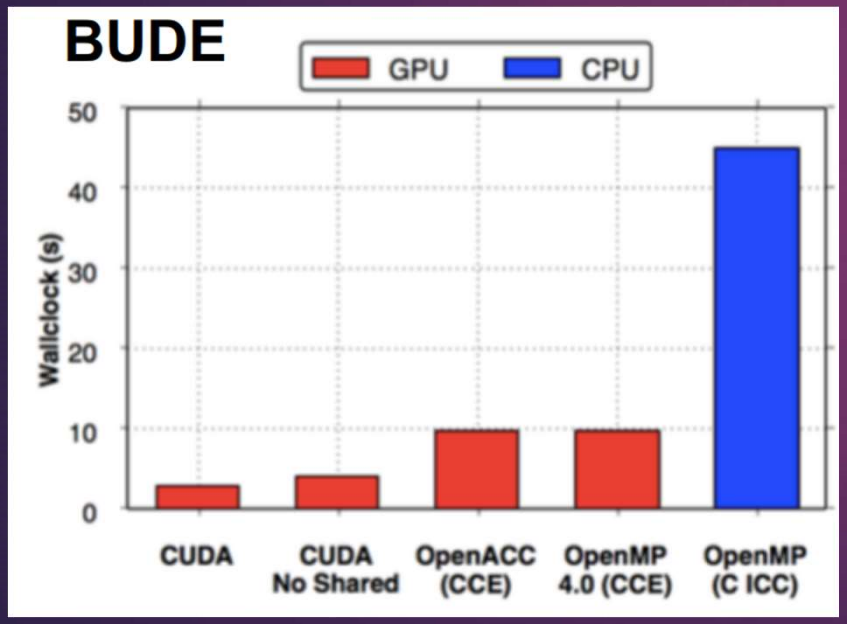
```
// Always evaluate expression
// and mask result
temp = (a - b*c);
mask = (a > b ? 1.f : 0.f);
acc += (mask * temp);
```

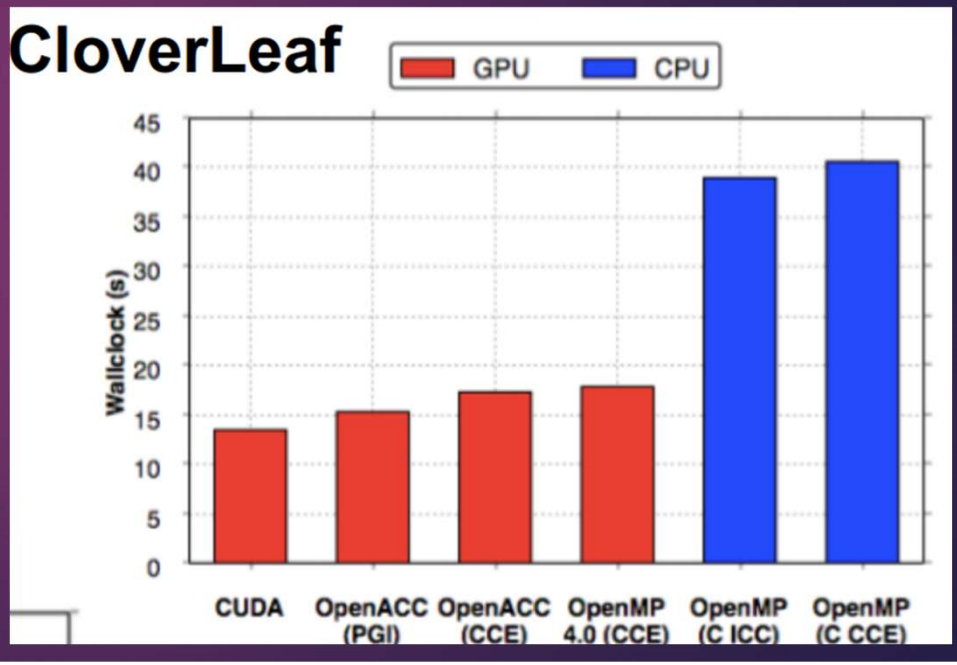
GPGPU OPENMP - COMPILERS

- **Intel** began support for OpenMP 4.0 targeting their Intel Xeon Phi coprocessors in 2013 (compiler version 15.0). Compiler version 17.0 and later versions support OpenMP 4.5
- **Cray** provided the first vendor supported implementation targeting NVIDIA GPUs late 2015. The latest version of CCE now supports all of OpenMP 4.5.
- **IBM** has recently completed a compiler implementation using Clang, that fully supports OpenMP 4.5. This is being introduced into the Clang main trunk.
- **GCC 6.1** introduced support for OpenMP 4.5, and can target Intel Xeon Phi, or HSA-enabled AMD GPUs. V7 added support for NVIDIA GPUs.
- **PGI** compilers don't currently support OpenMP on GPUs (but they do for CPUs).

CPU,GPGPU - OPENMP,OPENACC – ΧΡΟΝΟΣ ΥΠΟΛΟΓΙΣΜΩΝ (2016)

* CCE 8.4.3, ICC 15.0.3, PGI 15.01, CUDA 7.0 on an NVIDIA® K20X, and Intel® Xeon® Haswell 16 Core Processor (E5-2698 v3 @ 2.30GHz)





OPENACC ΚΑΙ GPGPU

Βασικές Οδηγίες

Για αρχάριους
#pragma acc kernels
Δηλώσεις const, restrict, __restrict

Για προχωρημένους
#pragma acc parallel loop
#pragma acc parallel loop private(…)
#pragma acc parallel loop reduction(+:sum))

```
#pragma acc parallel loop
for( int j = 0; j < N; j++) {
#pragma acc loop
      for( int i = 1; i < m-1; i++ ) {
```

```
#pragma acc parallel loop
      for(int i=0;i<N;i++) {
#pragma acc atomic update   read write
A[q[i+1] *=2 ;
```

Μεταφορές Δεδομένων

```
#pragma acc data pcreate(A[0:N])
pcopyout(B[0:N])
{
#pragma acc parallel loop
….
}
```

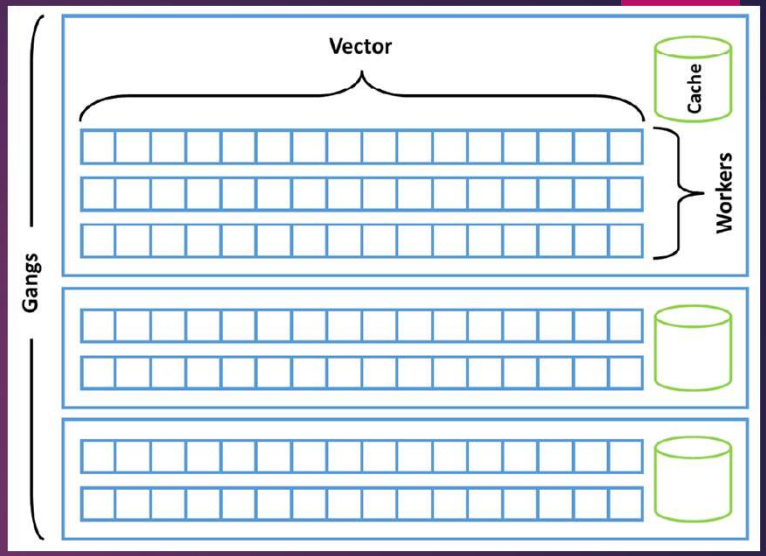
#pragma acc parallel copy(…) copyin(…)
copyout(…) create(…) present(…)

#pragma acc parallel default(none ή present)

OPENACC ΚΑΙ GPGPU

Ανανέωση Δεδομένων
CPU->GPU
#pragma acc update self(A[0:N])
GPU->CPU
#pragma acc update device(A[0:N])

Μεταφορές Δεδομένων σε αυθαίρετο χρόνο
#pragma acc enter data create(…) copyin(…)
#pragma acc exit data delete(...) copyout(...)




```
#pragma acc parallel loop gang
for ( i=0; i<N; i++)
#pragma acc loop worker vector(128)
      for ( j=0; j<M; j++)
```

```
#pragma acc parallel loop gang vector_length(128)
for ( i=0; i<N; i++)
#pragma acc loop vector
      for ( j=0; j<M; j++)
```

```
#pragma acc parallel loop num_gangs(n),
num_workers(n), vector_length(n)
```

OPENACC GPGPU

```
#pragma acc parallel loop
collapse(2)
for(int i=0; i<N; i++)
       for(int j=0; j<M; j++)

#pragma acc loop tile(8,8)
for(int i = 1; i <= N; i++)
for(int j = 1; j <= M; j++)
     a[i][j] = b[i+1][j]+b[i-
1][j]+b[i][j+1]+b[i][j-1]);

#pragma acc parallel loop async(1)
...
#pragma acc parallel loop async(1)
for(int i=0; i<N; i++)

#pragma acc wait(1)
```

Βελτιστοποίηση χρήσης μνήμης

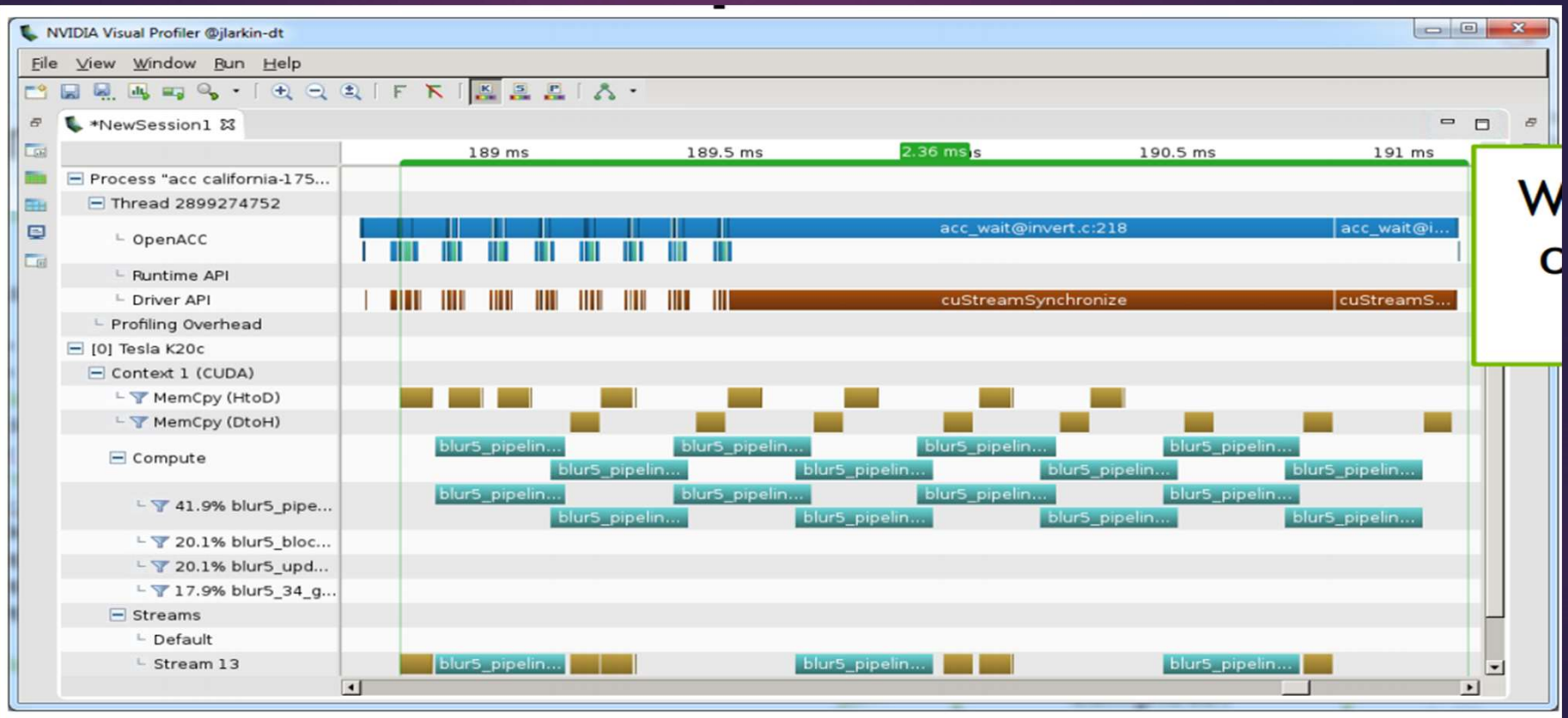
```
#pragma acc parallel loop device_type(nvidia) tile(16,8)
```

Συναρτήσεις στην openACC

```
#pragma acc routine seq    gang worker vector
  double f(int i) ;

#pragma acc parallel loop copy(A[0:N)
for(int i=0;i<N;i++)
     A[i] = f(i);
```

OPENACC GPGPU



OPENMP ΚΑΙ OPENACC: REDUCTION

Μεταβλητές

Char
Unsigned char
Int
Unsigned int
Float
Double

Operation	Effect	Language(s)
+	Sum	Fortran/C(++)
*	Product	Fortran/C(++)
max	Maximum	Fortran/C(++)
min	Maximum	Fortran/C(++)
&	Bitwise and	C(++)
|	Bitwise or	C(++)
&&	Logical and	C(++)
||	Logical or	C(++)