Legolas++ : automatic vectorisation for tensor-based algorithms

Software Development Workshop for Enterprise, HPC and AI
Tue 10th & Wed 11th December 2019

laurent.plagne@triscale-innov.com
kavoos.bojnourdi@triscale-innov.com
triscale-innov.com

TRISCALE innov

Purpose : Apply 1 algorithm to N problems

LEGOLAS++ Arrays

Solving N Tridiagonal systems

Under the hood

Performances !

Purpose : Apply 1 algorithm to N problems

LEGOLAS++ Arrays

Solving N Tridiagonal systems

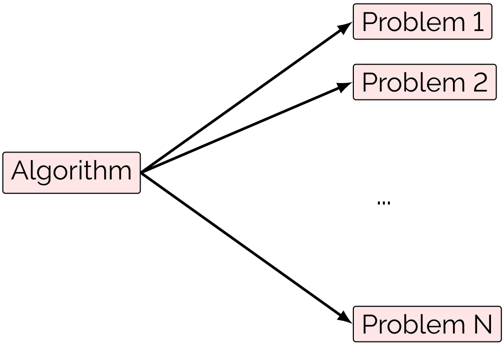
Under the hood

Performances !

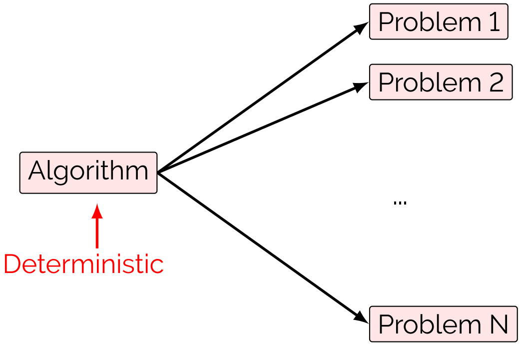
Considered Class of Problems :

Algorithm

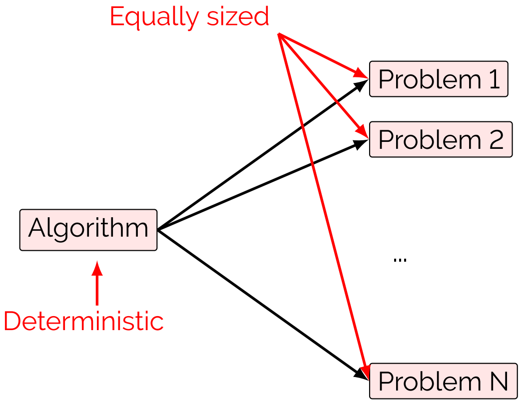
Considered Class of Problems :



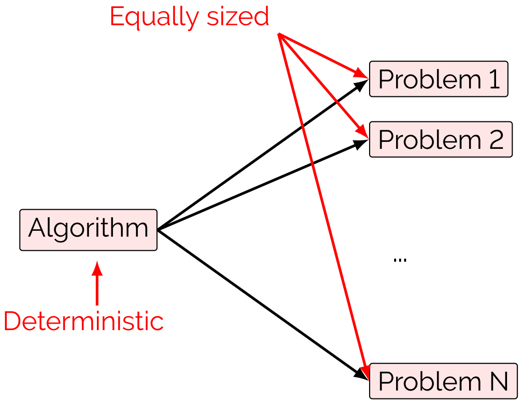
Considered Class of Problems :



Considered Class of Problems :

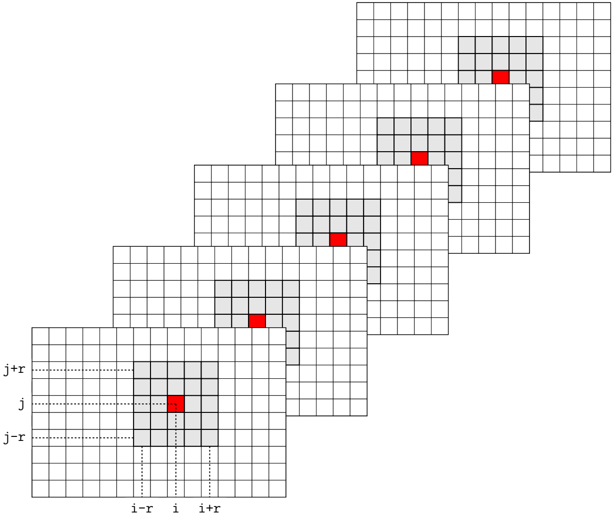


Considered Class of Problems :



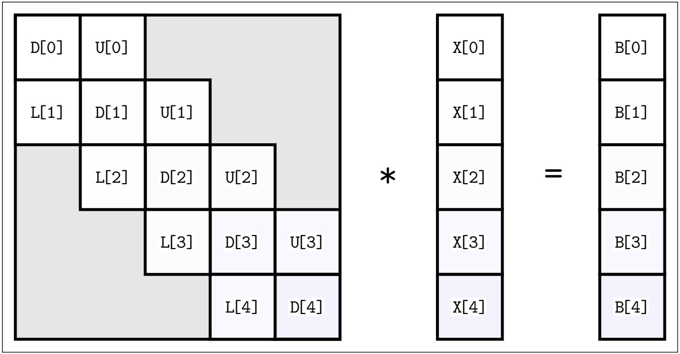
→ Vectorised and parallel Implementation

Example : Convolution Filter

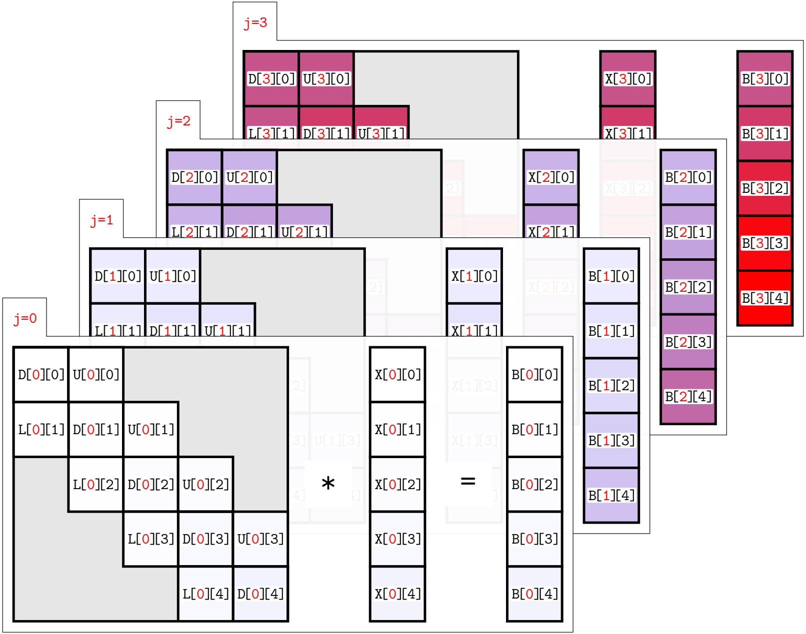


Running Example : Tridiagonal Linear Systems

Thomas Algorithm for $TX = B$:

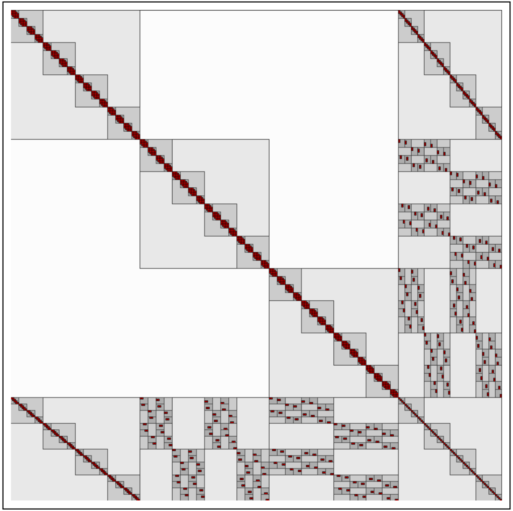


$\rightarrow$ algo : Thomas(`L`,`D`,`U`,`X`,`B`)


j=3
D[3][0] U[3][0]
L[3][1] D[3][1] U[3][1]
X[3][0]
X[3][1]
B[3][0]
B[3][1]
B[3][2]
B[3][3]
B[3][4]
j=2
D[2][0] U[2][0]
L[2][1] D[2][1] U[2][1]
X[2][0]
X[2][1]
B[2][0]
B[2][1]
B[2][2]
B[2][3]
B[2][4]
j=1
D[1][0] U[1][0]
L[1][1] D[1][1] U[1][1]
X[1][0]
X[1][1]
B[1][0]
B[1][1]
B[1][2]
B[1][3]
B[1][4]
j=0
D[0][0] U[0][0]
L[0][1] D[0][1] U[0][1]
L[0][2] D[0][2] U[0][2]
L[0][3] D[0][3] U[0][3]
L[0][4] D[0][4]
*
X[0][0]
X[0][1]
X[0][2]
X[0][3]
X[0][4]
=
B[0][0]
B[0][1]
B[0][2]
B[0][3]
B[0][4]

LEGOLAS++ library

LEGOLAS++ : building blocks for linear algebra solvers.
Highly structured sparse linear algebra.



LEGOLAS++ `Array<T,D>` Template class

Rectangular N-Dimensional Arrays (Tensors) :

```
//3D Array containing 100 double elts
Legolas::Array<double,3> X3D(10,5,2);
```

LEGOLAS++ Array<T,D> Template class

Rectangular N-Dimensional Arrays (Tensors) :

```
//3D Array containing 100 double elts
Legolas::Array<double,3> X3D(10,5,2);

Legolas::Array<double,2> X2D=X3D[0];

for (int k=0; k<10; k++)
 for (int j=0; j<5; j++)
  for (int i=0; i<2; i++)
   X3D[k][j][i]=2.*X3D[k][j][i]+1.;
```

Legolas++ Parallel Expression Template

```
//3D Array containing 100 double elts
Legolas::Array<double,3> X(10,5,2);
Legolas::Array<double,3> Y(10,5,2);
Legolas::Array<double,3> Z(10,5,2);


for (int k=0; k<10; k++)
 for (int j=0; j<5; j++)
  for (int i=0; i<2; i++)
  Y[k][j][i]+=2.*X[k][j][i]+Z[k][j][i];
```

```
Y+=2.*X+Z; //Expression Template
Y+=2.*X+Z || seq; // Sequential
Y+=2.*X+Z || par; // MultiThreaded
```

Legolas++ `Array<T,D>` Nested Types

`Array<T,D>` Nested Types	
`Scalar`	`Array<T,D>::Scalar` $\hat{=}$ `T`
`Element`	`Array<T,D>::Element` $\hat{=}$ `Array<T,D-1>` if $D > 1$ $\hat{=}$ `T&` if $D = 1$

Purpose : Apply 1 algorithm to N problems

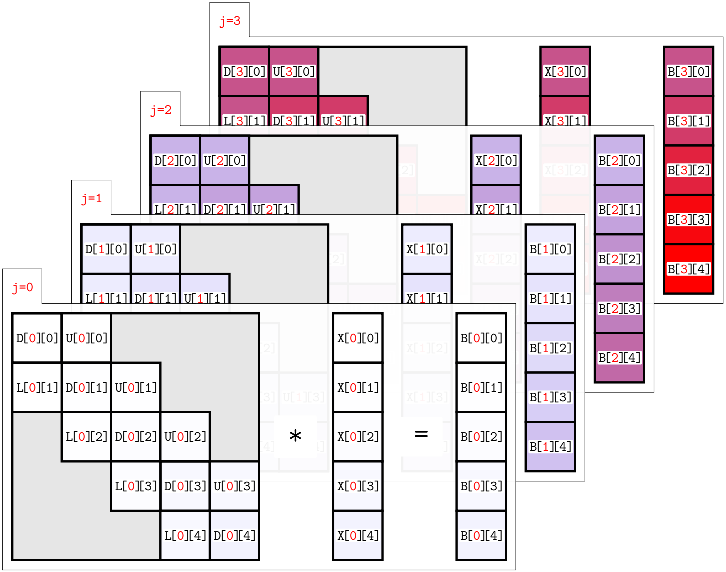
LEGOLAS++ Arrays

Solving N Tridiagonal systems

Under the hood

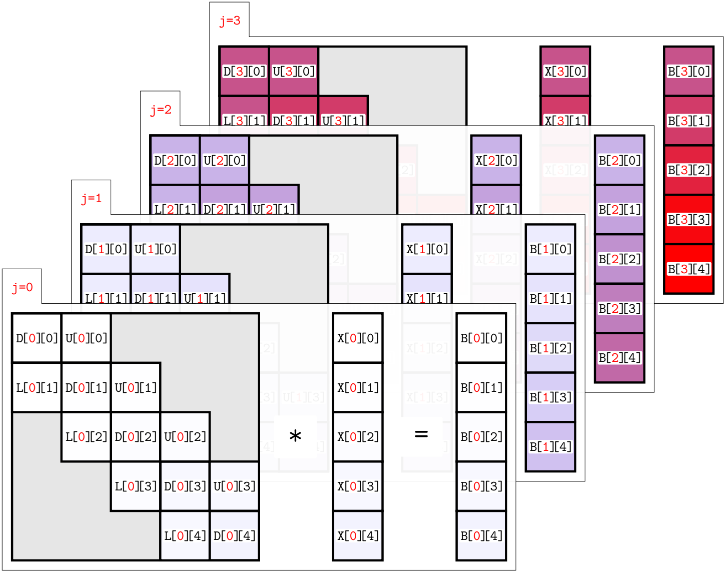
Performances !

Legolas::Array<float,2> L,D,U,X,B   (L,D,U)[j][i]



→ algo : Thomas

Legolas::Array<float,2> L,D,U,X,B   (L,D,U)[j][i]



→ algo : Thomas(L[j],D[j],U[j],X[j],B[j])

Solving *N* Tridiagonal systems

```
struct MultiThomasSolver{
  template <class A2D>
  void operator()(int begin, int end,
                  A2D D2D, A2D U2D, A2D L2D
                  A2D B2D, A2D X2D) const{
    using Element=typename A2D::Element;
    using Scalar =typename A2D::Scalar;
    Element S(X2D[0].shape());
    //Loop on tridiagonal systems
    for (int j=begin ; j<end ; j++){
      auto D=D2D[j]; auto U=U2D[j]; auto L=L2D[j];
      auto B=B2D[j]; auto X=X2D[j];
      //Thomas Algorithm starts here

       //forward...
       //backward...

    }
}};
```

Thomas Algorithm : TX=B with T=(L,D,U)

```
Scalar s=D[0],one=1.0,sm1=one/s;
const int size=X.size();
//forward
X[0]=B[0]*sm1;
for(int i=1; i<size; i++){
  S[i]=U[i-1]*sm1;
  s=D[i]-L[i]*S[i];
  X[i]=B[i]-L[i]*X[i-1];
  sm1=one/s;
  X[i]*=sm1;
}
//backward
for (int i=(size-2);i>=0 ; i--){
  X[i]-=S[i+1]*X[i+1];
};
```

Legolas++ map

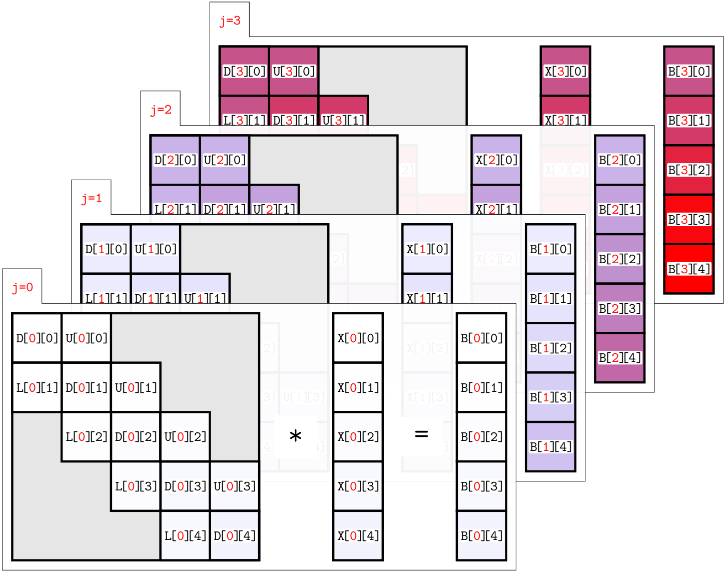
```cpp
int main () {

  size_t ni=200;//System size
  size_t nj=800;//Number of systems

  using A2D=Legolas::Array<float,2>;

  A2D u(nj,ni),l(nj,ni),d(nj,ni);
  A2D X(nj,ni),B(nj,ni);
  //.. Arrays initialization
  ...
   //Solve all systems (sequential)
  Legolas::map(MultiThomasSolver(),d,u,l,B,X);

}
```

SIMD Parallelism

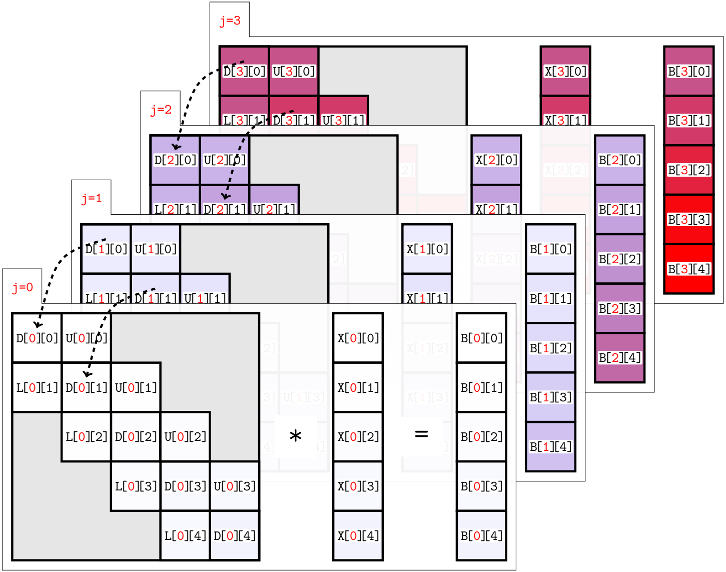
SIMD : Single Instruction Multiple Data
→ CPU (MMX,SSE,AVX,AVX2,AVX512,...)

→ OK for contiguous data !

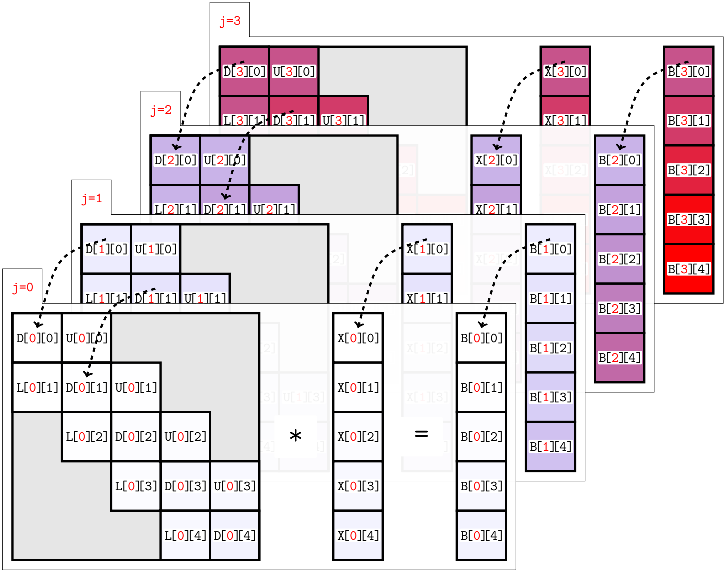
Vectorisation with SIMD width=2



Vectorisation with SIMD width=2



Vectorisation with SIMD width=2



LEGOLAS++ : Data Layout Interleaving

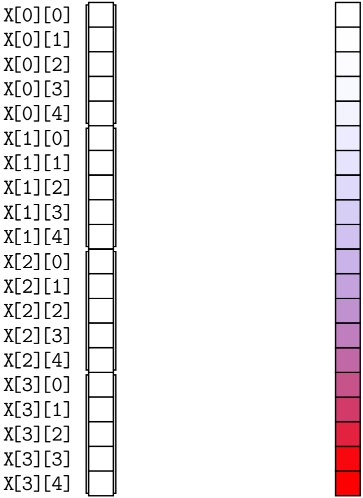
The LEGOLAS++ `Array` template class :

`Legolas::Array<T,D,P,DP>`

accepts two extra parameters `P` and `DP` that control multi-dimensional array data layout.
`P` specifies the *packing factor* that must be applied to the dimension dimension `DP`.

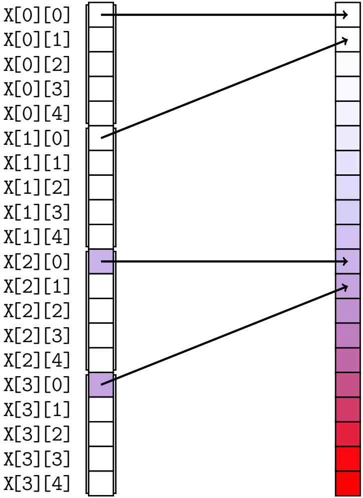
Vectorisation → Data Layout Interleaving

```
Legolas::Array<float,2,2,2> X(nj,ni)
```



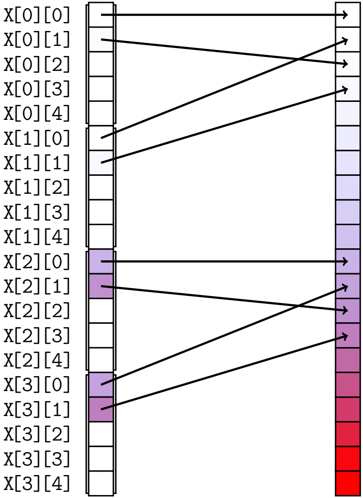
Vectorisation → Data Layout Interleaving

`Legolas::Array<float,2,2,2> X(nj,ni)`



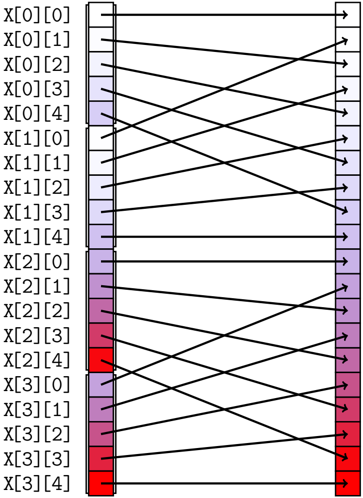
Vectorisation → Data Layout Interleaving

```
Legolas::Array<float,2,2,2> X(nj,ni)
```



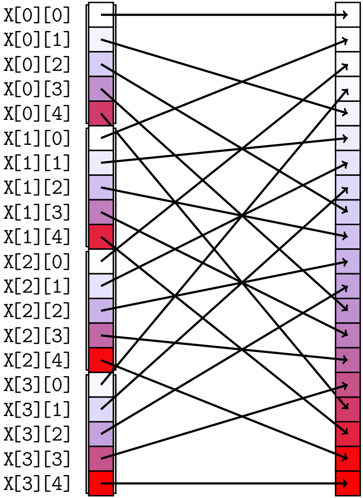
Vectorisation → Data Layout Interleaving

`Legolas::Array<float,2,2,2> X(nj,ni)`



Vectorisation → Data Layout Interleaving

```
Legolas::Array<float,2,4,2> X(nj,ni)
```



Legolas++ for N tridiagonal systems

```
int main () {
  size_t ni=200;//System size
  size_t nj=800;//Number of systems

  using A2D=Legolas::Array<float,2>;



  A2D u(nj,ni),l(nj,ni),d(nj,ni);
  A2D X(nj,ni),B(nj,ni);
  //.. Arrays initialization
  ...
  //Solve all systems (sequential)
  Legolas::map(MultiThomasSolver(),d,u,l,B,X);


}
```

LEGOLAS++ for *N* tridiagonal systems

```
int main () {
  size_t ni=200;//System size
  size_t nj=800;//Number of systems

  //using A2D=Legolas::Array<float,2>;
  using A2D=Legolas::Array<float,2,4,2>;


  A2D u(nj,ni),l(nj,ni),d(nj,ni);
  A2D X(nj,ni),B(nj,ni);
  //.. Arrays initialization
  ...
  //Automatic SSE Vectorization
  Legolas::map(MultiThomasSolver(),d,u,l,B,X);


}
```

LEGOLAS++ for *N* tridiagonal systems

```cpp
int main () {
  size_t ni=200;//System size
  size_t nj=800;//Number of systems

  //using A2D=Legolas::Array<float,2>;
  //using A2D=Legolas::Array<float,2,4,2>;
  using A2D=Legolas::Array<float,2,8,2>;

  A2D u(nj,ni),l(nj,ni),d(nj,ni);
  A2D X(nj,ni),B(nj,ni);
  //.. Arrays initialization
  ...
  //Automatic AVX Vectorization
  Legolas::map(MultiThomasSolver(),d,u,l,B,X);


}
```

LEGOLAS++ for N tridiagonal systems

```
int main () {
  size_t ni=200;//System size
  size_t nj=800;//Number of systems

  //using A2D=Legolas::Array<float,2>;
  //using A2D=Legolas::Array<float,2,4,2>;
  using A2D=Legolas::Array<float,2,8,2>;

  A2D u(nj,ni),l(nj,ni),d(nj,ni);
  A2D X(nj,ni),B(nj,ni);
  //.. Arrays initialization
  ...
  //Automatic AVX Vectorization+parallelization
  //Legolas::map(MultiThomasSolver(),d,u,l,B,X);
  Legolas::parmap(MultiThomasSolver(),d,u,l,B,X);

}
```

Generic template C++ libraries combination

LEGOLAS++

Generic template C++ libraries combination

Legolas++

- controls Data Layout Interleaving : `Legolas::Array<T,D,P,DP>`

Generic template C++ libraries combination

LEGOLAS++

- controls Data Layout Interleaving : `Legolas::Array<T,D,P,DP>`
- uses Eigen for SIMD operations :
  `http://eigen.tuxfamily.org`

Generic template C++ libraries combination

Legolas++

- controls Data Layout Interleaving : `Legolas::Array<T,D,P,DP>`
- uses Eigen for SIMD operations : `http://eigen.tuxfamily.org`
- uses Intel TBB for *multithreading* : `https://www.threadingbuildingblocks.org`

Generic template C++ libraries combination

LEGOLAS++

- controls Data Layout Interleaving : `Legolas::Array<T,D,P,DP>`
- uses Eigen for SIMD operations : `http://eigen.tuxfamily.org`
- uses Intel TBB for *multithreading* : `https://www.threadingbuildingblocks.org`
- $\rightarrow$ *header-only* library (<2K lines)!

Eigen Blocks (fixed size arrays)

```
typedef Eigen::Array<float,p,1> Block;
```

Eigen overloads all the usual arithmetic operators, functions and expressions.
For example :

```
Block a,b,c,d;
a+=b-c;    //_mm_add_ps
a=b*c/d;   //_mm_mul_ps,_mm_div_ps
```

are transformed, at compile time, into the corresponding explicit intrinsic SIMD functions without any performance penalty compared to hand-coded assembly or intrinsic formulation.

LEGOLAS++ getPackedView()

- Let us define X as :

```
Legolas::Array<float,2,4,2> X(nj,ni);
```

LEGOLAS++ getPackedView()

- Let us define X as :

```
Legolas::Array<float,2,4,2> X(nj,ni);
```

- and Block as :

```
using Block=Eigen::Array<float,4,1>;
```

Legolas++ getPackedView()

- Let us define `X` as :

```
Legolas::Array<float,2,4,2> X(nj,ni);
```

- and `Block` as :

```
using Block=Eigen::Array<float,4,1>;
```

- Then `X` can be reinterpreted as non-interleaved array of `Block`s :

LEGOLAS++ getPackedView()

- Let us define X as :
  ```
  Legolas::Array<float,2,4,2> X(nj,ni);
  ```
- and Block as :
  ```
  using Block=Eigen::Array<float,4,1>;
  ```
- Then X can be reinterpreted as non-interleaved array of Blocks :
  - ```
    Legolas::Array<Block,2>
    ```
    ```

LEGOLAS++ `getPackedView()`

- Let us define `X` as :
  ```
  Legolas::Array<float,2,4,2> X(nj,ni);
  ```
- and `Block` as :
  ```
  using Block=Eigen::Array<float,4,1>;
  ```
- Then `X` can be reinterpreted as non-interleaved array of `Block`s :
  - ```
    Legolas::Array<Block,2>
    ```
- LEGOLAS++ `getPackedView()` method returns this view :
    ```

LEGOLAS++ `getPackedView()`

- Let us define `X` as :

```
Legolas::Array<float,2,4,2> X(nj,ni);
```

- and `Block` as :

```
using Block=Eigen::Array<float,4,1>;
```

- Then `X` can be reinterpreted as non-interleaved array of `Block`s :
  - `Legolas::Array<Block,2>`
- LEGOLAS++ `getPackedView()` method returns this view :
  - `Legolas::Array<Block,2> Xp=X.getPackedView();`

Legolas++ map

```
template <class ALGO,typename... ARRAYS>
void map(ALGO a, ARRAYS... rest){

  int s=getArraysSize(rest...);//e.g 5
  int p=getPackSize(rest...);  //e.g 2
  int sp=getPackedArraysSize(rest...);// 5/2=2

  // Vectorized part:
  // algo a in [0,sp=s/p[ -> [0,2[
  a(0,sp,rest.getPackedView()...);

  // Scalar remainder
  // scalar algo a a in [sp*p,s[ ->[4,5[
  a(sp*p,s,rest...);
}
```

A2D= Array<Block,2>

```
struct MultiThomasSolver{
  template <class A2D>
  void operator()(int begin, int end,
                  A2D D2D, A2D U2D, A2D L2D
                  A2D B2D, A2D X2D) const{
    using Element=typename A2D::Element;
    using Scalar =typename A2D::Scalar;
    Element S(X2D[0].shape());
    Scalar  one(1.0),s,sm1;
    //Loop on tridiagonal systems
    for (int j=begin ; j<end ; j++){
      auto D=D2D[j]; auto U=U2D[j]; auto L=L2D[j];
      auto B=B2D[j]; auto X=X2D[j];
      const int size=X.size();
      X[0]=B[0]*sm1;
      for(int i=1; i<size; i++){
        S[i]=U[i-1]*sm1;
        ...
```

A2D= Array<Block,2>

```
struct MultiThomasSolver{
  template <class A2D>
  void operator()(int begin, int end,
                  A2D D2D, A2D U2D, A2D L2D
                  A2D B2D, A2D X2D) const{
    using Element=typename A2D::Element;//Array<Block,1>
    using Scalar =typename A2D::Scalar;
    Element S(X2D[0].shape());
    Scalar  one(1.0),s,sm1;
    //Loop on tridiagonal systems
    for (int j=begin ; j<end ; j++){
      auto D=D2D[j]; auto U=U2D[j]; auto L=L2D[j];
      auto B=B2D[j]; auto X=X2D[j];
      const int size=X.size();
      X[0]=B[0]*sm1;
      for(int i=1; i<size; i++){
        S[i]=U[i-1]*sm1;
        ...
```

A2D= Array<Block,2>

```
struct MultiThomasSolver{
  template <class A2D>
  void operator()(int begin, int end,
                  A2D D2D, A2D U2D, A2D L2D
                  A2D B2D, A2D X2D) const{
    using Element=typename A2D::Element;//Array<Block,1>
    using Scalar =typename A2D::Scalar; //Block
    Element S(X2D[0].shape());
    Scalar  one(1.0),s,sm1;
    //Loop on tridiagonal systems
    for (int j=begin ; j<end ; j++){
      auto D=D2D[j]; auto U=U2D[j]; auto L=L2D[j];
      auto B=B2D[j]; auto X=X2D[j];
      const int size=X.size();
      X[0]=B[0]*sm1;
      for(int i=1; i<size; i++){
        S[i]=U[i-1]*sm1;
        ...
```

A2D= Array<Block,2>

```
struct MultiThomasSolver{
  template <class A2D>
  void operator()(int begin, int end,
                  A2D D2D, A2D U2D, A2D L2D
                  A2D B2D, A2D X2D) const{
    using Element=typename A2D::Element;//Array<Block,1>
    using Scalar =typename A2D::Scalar; //Block
    Element S(X2D[0].shape());
    Scalar  one(1.0),s,sm1;
    //Loop on tridiagonal systems
    for (int j=begin ; j<end ; j++){
      auto D=D2D[j]; auto U=U2D[j]; auto L=L2D[j];
      auto B=B2D[j]; auto X=X2D[j];
      const int size=X.size();
      X[0]=B[0]*sm1;//_mm_mul_ps
      for(int i=1; i<size; i++){
        S[i]=U[i-1]*sm1;//_mm_mul_ps
        ...
```

LEGOLAS++ parmap

```
template <class ALGO,typename... ARRAYS>
void parmap(int begin, int end,
ALGO algo, ARRAYS... rest){

  tbb::parallel_for(
  tbb::blocked_range<int>(begin,end),
  [=](tbb::blocked_range<int> r){
    algo(r.begin(),r.end(),rest...);},
  tbb::auto_partitioner());

}
```

Purpose : Apply 1 algorithm to N problems

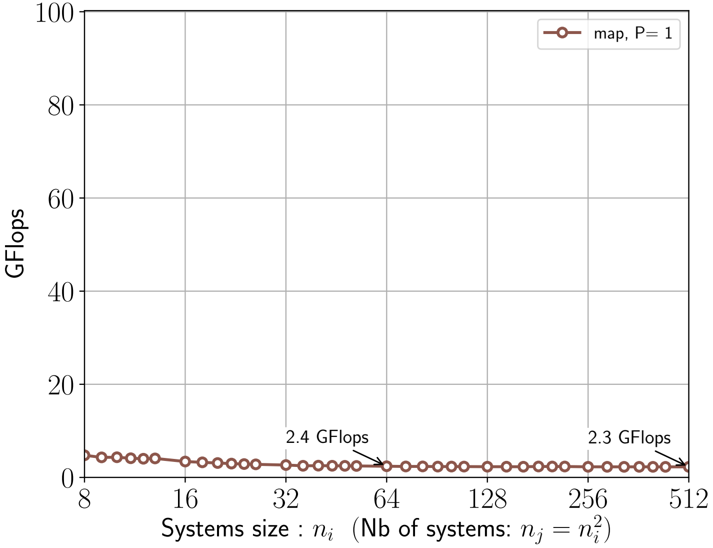
Legolas++ Arrays

Solving N Tridiagonal systems

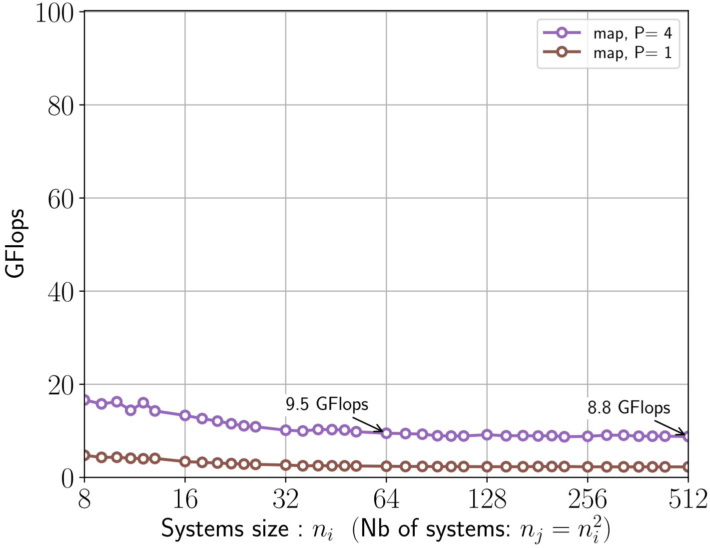
Under the hood

Performances !

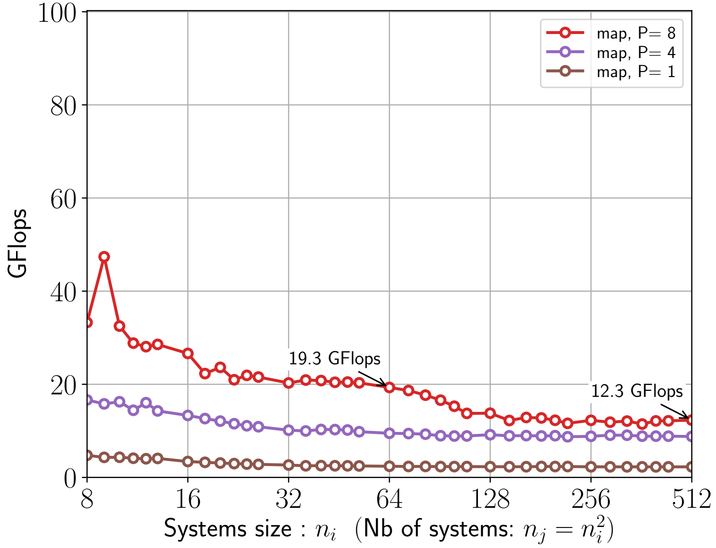
MultiThomas (Skylake 4-core,4GHz,AVX2)



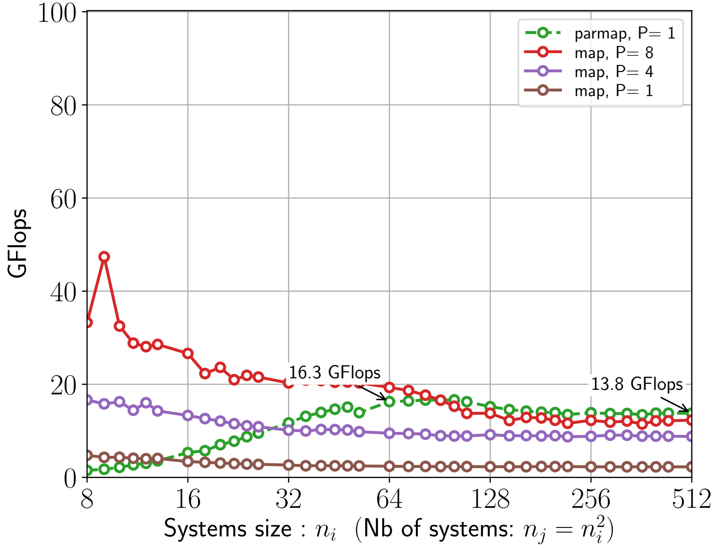
MultiThomas (Skylake 4-core,4GHz,AVX2)



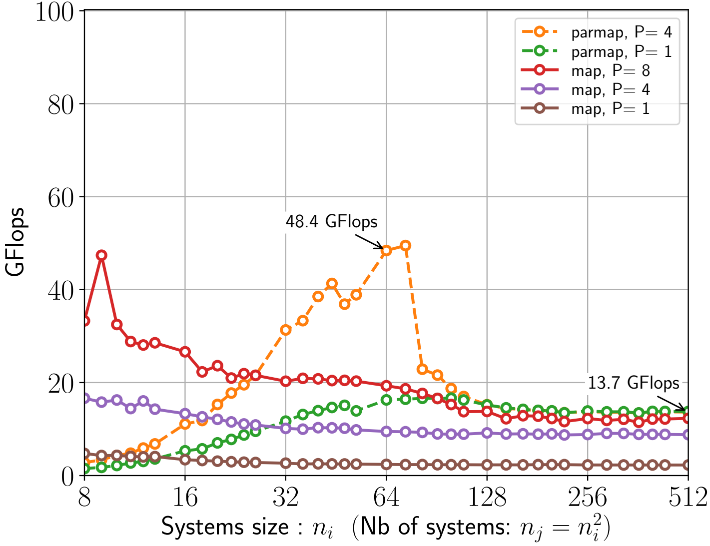
MultiThomas (Skylake 4-core,4GHz,AVX2)



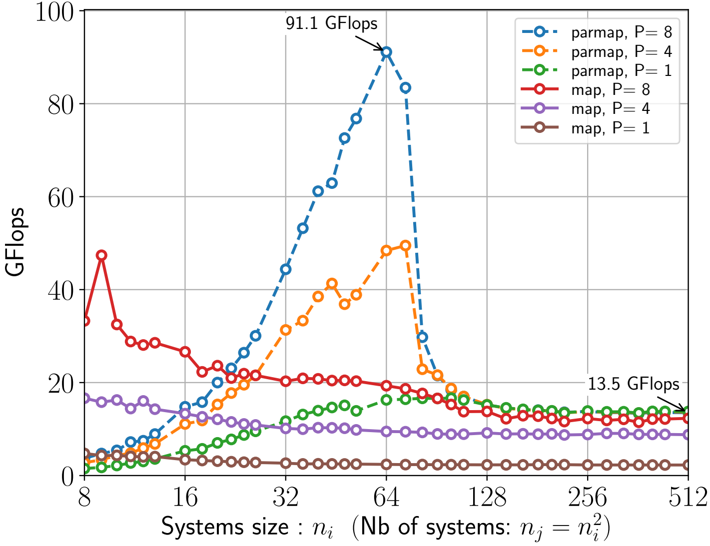
MultiThomas (Skylake 4-core,4GHz,AVX2)

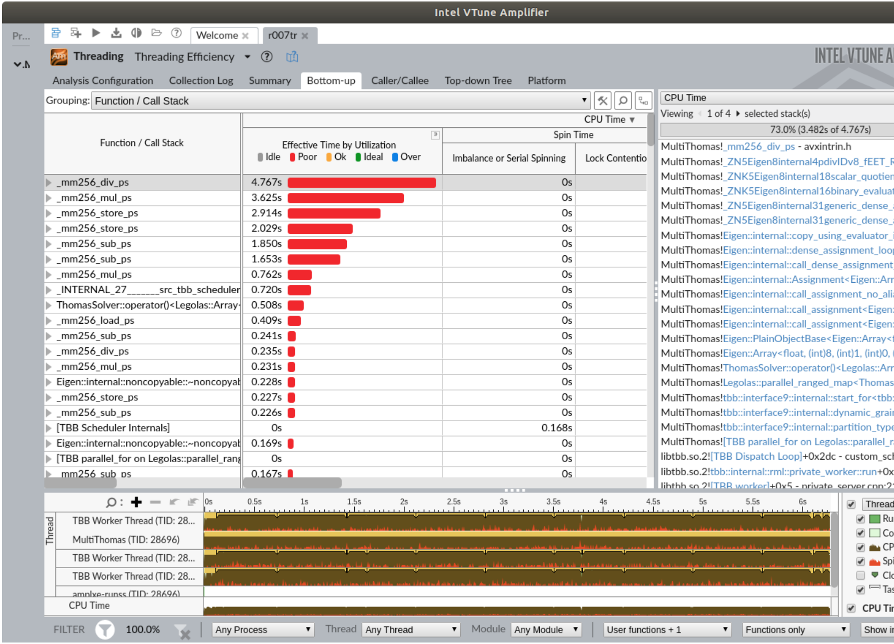


MultiThomas (Skylake 4-core,4GHz,AVX2)



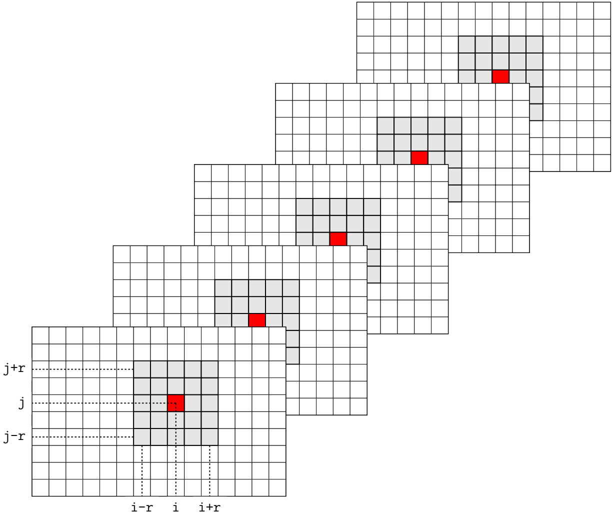
MultiThomas (Skylake 4-core,4GHz,AVX2)



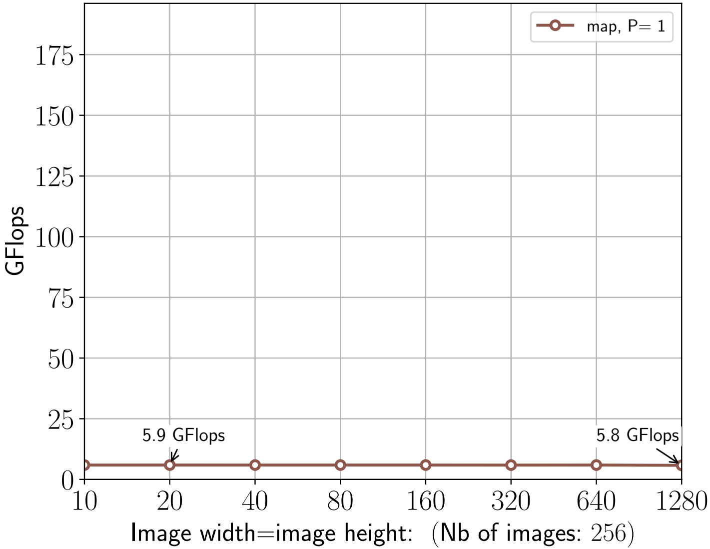

Intel VTune Amplifier
Welcome
r007tr
Threading
Threading Efficiency
Analysis Configuration
Collection Log
Summary
Bottom-up
Caller/Callee
Top-down Tree
Platform
Grouping: Function / Call Stack
Function / Call Stack
CPU Time
Effective Time by Utilization
Idle Poor Ok Ideal Over
Spin Time
Imbalance or Serial Spinning
_mm256_div_ps 4.767s 0s
_mm256_mul_ps 3.625s 0s
_mm256_store_ps 2.914s 0s
_mm256_store_ps 2.029s 0s
_mm256_sub_ps 1.850s 0s
_mm256_sub_ps 1.653s 0s
_mm256_mul_ps 0.762s 0s
0.720s 0s
0.508s 0s
_mm256_load_ps 0.409s 0s
_mm256_sub_ps 0.241s 0s
_mm256_div_ps 0.235s 0s
_mm256_mul_ps 0.231s 0s
0.228s 0s
_mm256_store_ps 0.227s 0s
_mm256_sub_ps 0.226s 0s
[TBB Scheduler Internals] 0s 0.168s
0.169s 0s
0s 0s
Viewing 1 of 4 selected stack(s)
73.0% (3.482s of 4.767s)
MultiThomas!_mm256_div_ps - avxintrin.h
TBB Worker Thread (TID: 28...
MultiThomas (TID: 28696)
TBB Worker Thread (TID: 28...
TBB Worker Thread (TID: 28...
Thread
CPU Time
0s 0.5s 1s 1.5s 2s 2.5s 3s 3.5s 4s 4.5s 5s 5.5s 6s
FILTER 100.0%
Any Process
Thread
Any Thread
Module
Any Module
User functions + 1
Functions only

Example : Convolution Image Filter

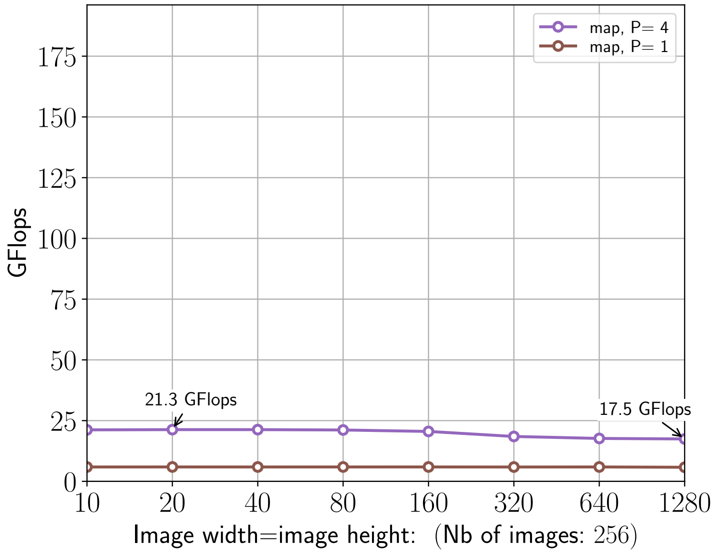
$$\text{convolution}(\{(s_0, t_0), (s_1, t_1) \ldots, (s_n, t_n)\})$$



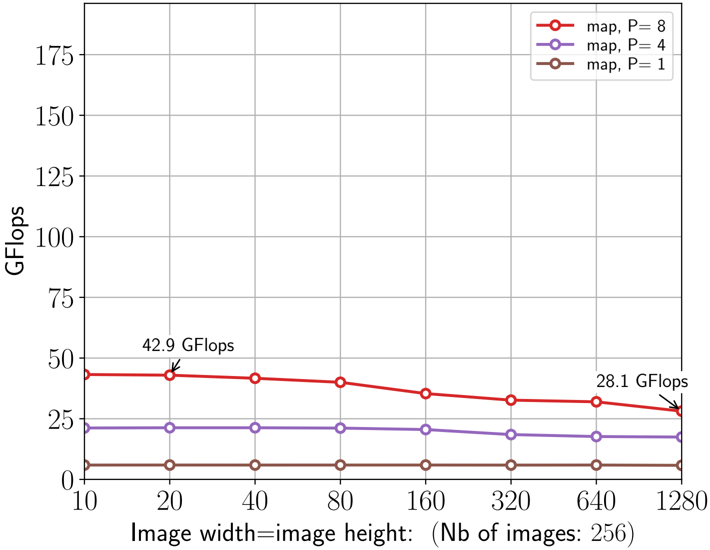
Legolas++ Multiple Image Convolution Filter (Skylake 4-core,4GHz,AVX2)



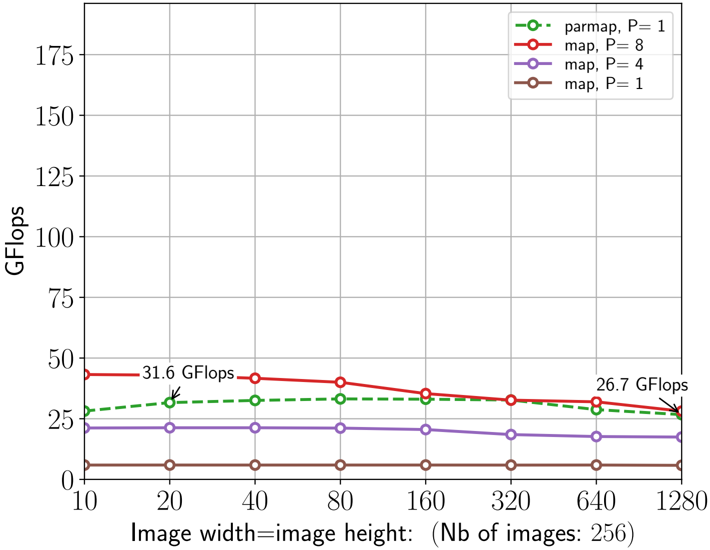
LEGOLAS++ Multiple Image Convolution Filter (Skylake 4-core,4GHz,AVX2)



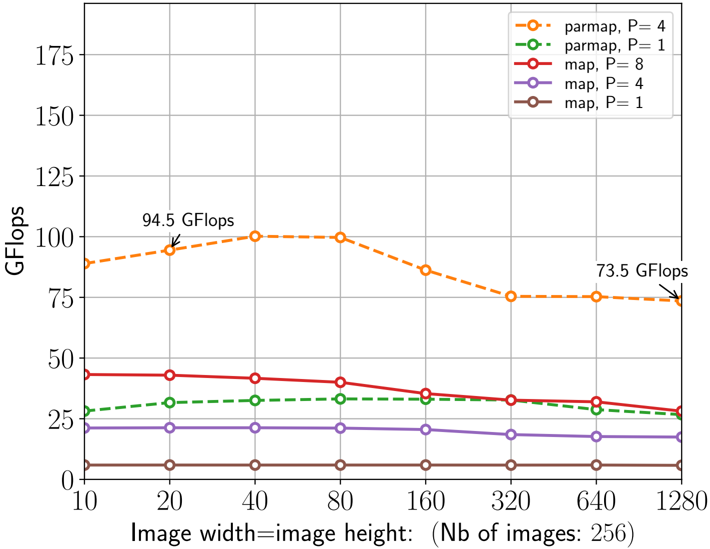
LEGOLAS++ Multiple Image Convolution Filter (Skylake 4-core,4GHz,AVX2)



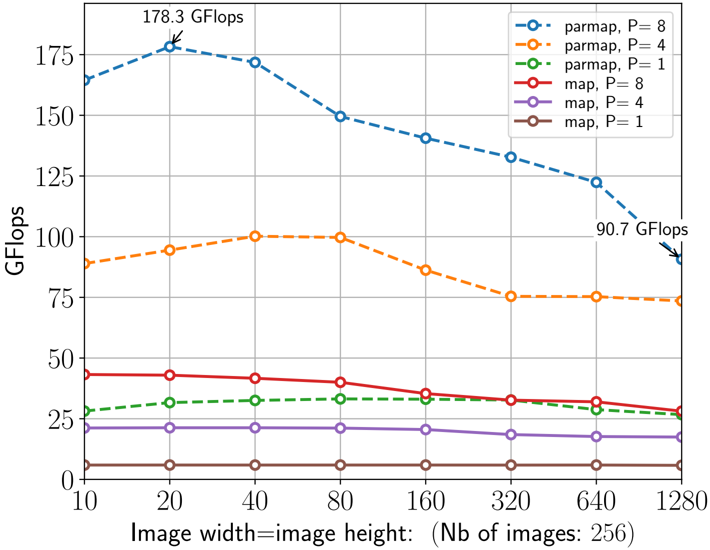
LEGOLAS++ Multiple Image Convolution Filter (Skylake 4-core,4GHz,AVX2)



LEGOLAS++ Multiple Image Convolution Filter (Skylake 4-core,4GHz,AVX2)



LEGOLAS++ Multiple Image Convolution Filter (Skylake 4-core,4GHz,AVX2)



Conclusion

- Considering the

Conclusion

- Considering the
  - multiple applications of a deterministic algorithm

Conclusion

- Considering the
  - multiple applications of a deterministic algorithm
  - to equally sized problems,

Conclusion

- Considering the
  - multiple applications of a deterministic algorithm
  - to equally sized problems,
- `Legolas::Array<T,D,P,DP>` enables for automatic

Conclusion

- Considering the
  - multiple applications of a deterministic algorithm
  - to equally sized problems,
- `Legolas::Array<T,D,P,DP>` enables for automatic
  - vectorisation via Data Layout Interleaving (+Eigen)

Conclusion

- Considering the
  - multiple applications of a deterministic algorithm
  - to equally sized problems,
- `Legolas::Array<T,D,P,DP>` enables for automatic
  - vectorisation via Data Layout Interleaving (+Eigen)
  - parallelisation (Intel TBB)

Conclusion

- Considering the
  - multiple applications of a deterministic algorithm
  - to equally sized problems,
- `Legolas::Array<T,D,P,DP>` enables for automatic
  - vectorisation via Data Layout Interleaving (+Eigen)
  - parallelisation (Intel TBB)
  - from a single user expression of algorithms.

Thank you for your attention !

TriScale innov concentrates advanced expertise on **3 axis** :

TriScale innov concentrates advanced expertise on 3 **axis** :

- Applied Mathematics (Simulation, Mathematical Optimisation...);

TriScale innov concentrates advanced expertise on **3 axis** :

- Applied Mathematics (Simulation, Mathematical Optimisation...);
- Software Development (Julia, Rust, Go, C, C++, Java...);

TriScale innov concentrates advanced expertise on **3 axis** :

- Applied Mathematics (Simulation, Mathematical Optimisation...);
- Software Development (Julia, Rust, Go, C, C++, Java...);
- Hardware Architectures (SMP Clusters, GPU, DSPs, IoT...).

TriScale innov concentrates advanced expertise on **3 axis** :

- Applied Mathematics (Simulation, Mathematical Optimisation...);
- Software Development (Julia, Rust, Go, C, C++, Java...);
- Hardware Architectures (SMP Clusters, GPU, DSPs, IoT...).

The success of scientific and technical software depends on the **optimal combination of these 3 areas in strong interaction**.

TriScale innov concentrates advanced expertise on **3 axis** :

- Applied Mathematics (Simulation, Mathematical Optimisation…);
- Software Development (Julia, Rust, Go, C, C++, Java…);
- Hardware Architectures (SMP Clusters, GPU, DSPs, IoT…).

The success of scientific and technical software depends on the **optimal combination of these 3 areas in strong interaction**.

Training

- HPC & //ism
- Julia, Go
- Numerical Quality

TriScale innov concentrates advanced expertise on 3 **axis** :

- Applied Mathematics (Simulation, Mathematical Optimisation...);
- Software Development (Julia, Rust, Go, C, C++, Java...);
- Hardware Architectures (SMP Clusters, GPU, DSPs, IoT...).

The success of scientific and technical software depends on the **optimal combination of these 3 areas in strong interaction**.

Training

- HPC & //ism
- Julia, Go
- Numerical Quality

Consulting

- Architecture
- Performance
- Algorithms

TriScale innov concentrates advanced expertise on 3 **axis** :

- Applied Mathematics (Simulation, Mathematical Optimisation...);
- Software Development (Julia, Rust, Go, C, C++, Java...);
- Hardware Architectures (SMP Clusters, GPU, DSPs, IoT...).

The success of scientific and technical software depends on the **optimal combination of these 3 areas in strong interaction**.

Training

- HPC & //ism
- Julia, Go
- Numerical Quality

Consulting

- Architecture
- Performance
- Algorithms

Software

- Scientific Softwares
- Industrialisation of research prototypes

TriScale innov concentrates advanced expertise on **3 axis** :

- Applied Mathematics (Simulation, Mathematical Optimisation...);
- Software Development (Julia, Rust, Go, C, C++, Java...);
- Hardware Architectures (SMP Clusters, GPU, DSPs, IoT...).

The success of scientific and technical software depends on the **optimal combination of these 3 areas in strong interaction**.

Training

- HPC & //ism
- Julia, Go
- Numerical Quality

Consulting

- Architecture
- Performance
- Algorithms

Software

- Scientific Softwares
- Industrialisation of research prototypes

→ More information on triscale-innov.com

```cpp
int main(int argc, char** argv) {
  typedef float RealType;

  int nx = 800;       // width
  int ny = 600;       // height

  typedef Legolas::Array<RealType,4,8,4> A4D;
  const int nz = 200; // nimages
  const int nc = 4;   //nchannels(rgba)

  A4D sourceImages(nz, nc, ny, nx);
  A4D targetImages(nz, nc, ny, nx);

  //init source images

  typedef Convolution<GaussianKernel<5>> MyFilter;
  Legolas::map(MyFilter(),sourceImages,targetImages);

  return 0;
 }
```

```cpp
template <class KERNEL>
struct Convolution {
  template <class A4D>
  void operator()(int begin, int end,
  const A4D source, A4D target) const {
    typedef typename A4D::RealType Scalar;
  ...
    for (int k = begin; k < end; k++) {
      auto source3D = source[k];
      auto target3D = target[k];

      const int ny = source3D[0].size();
      const int nx = source3D[0][0].size();
      const int r = KERNEL::radius;
      const Scalar zero(0.0);

      Scalar sumR(zero),sumG(zero),sumB(zero);

      for (int j = 0; j < ny; j++) {
        for (int i = 0; i < nx; i++) {
          sumR = zero; sumG = zero; sumB = zero;
          for (int sj = -r; sj <= r; sj++) {
            int jp = j + sj; if (jp < 0 || jp >= ny) jp = j - sj;
            for (int si = -r; si <= r; si++) {
              int ip = i + si;  if (ip < 0 || ip >= nx) ip = i - si;

              sumR += kmat[sj + r][si + r] * source3D[0][jp][ip];
              sumG += kmat[sj + r][si + r] * source3D[1][jp][ip];
              sumB += kmat[sj + r][si + r] * source3D[2][jp][ip];
            }
          }

          target3D[0][j][i] = sumR;
          target3D[1][j][i] = sumG;
          target3D[2][j][i] = sumB;
        }}}}};
```

Considered Problem

Vectorize (and parallelize) multiple applications of *deterministic* algorithms to *regular* set of datasets.

Considered Problem

Vectorize (and parallelize) multiple applications of *deterministic* algorithms to *regular* set of datasets.

- Let $V = \{v_1, v_2, \ldots, v_n\}$ be a dataset consisting of n multidim-array variables.

Considered Problem

Vectorize (and parallelize) multiple applications of *deterministic* algorithms to *regular* set of datasets.

- Let $V = \{v_1, v_2, \ldots, v_n\}$ be a dataset consisting of n multidim-array variables.
- Let $a(V)$ be a *deterministic* algorithm to be applied to a dataset V :

Considered Problem

Vectorize (and parallelize) multiple applications of *deterministic* algorithms to *regular* set of datasets.

- Let $V = \{v_1, v_2, \ldots, v_n\}$ be a dataset consisting of n multidim-array variables.
- Let $a(V)$ be a *deterministic* algorithm to be applied to a dataset V :
  - The sequence of operations does not depend on the dataset.

Considered Problem

Vectorize (and parallelize) multiple applications of *deterministic* algorithms to *regular* set of datasets.

- Let $V = \{v_1, v_2, \ldots, v_n\}$ be a dataset consisting of n multidim-array variables.
- Let $a(V)$ be a *deterministic* algorithm to be applied to a dataset V :
  - The sequence of operations does not depend on the dataset.
- Let $W = \{V_1, V_2, \ldots, V_m\}$ be a set of m datasets. W is *regular* if :

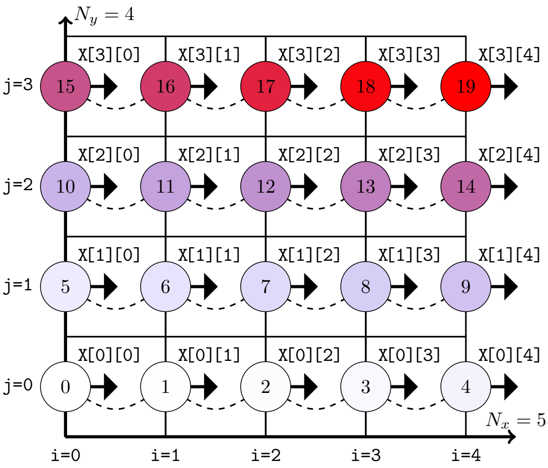
Considered Problem

Vectorize (and parallelize) multiple applications of *deterministic* algorithms to *regular* set of datasets.

- Let $V = \{v_1, v_2, \ldots, v_n\}$ be a dataset consisting of n multidim-array variables.
- Let $a(V)$ be a *deterministic* algorithm to be applied to a dataset V :
  - The sequence of operations does not depend on the dataset.
- Let $W = \{V_1, V_2, \ldots, V_m\}$ be a set of m datasets. W is *regular* if :
  - $\boxed{\forall j, j' \in [1, m]^2,\ \forall i \in [1, n], \quad \mathrm{size}(v_{i,j}) = \mathrm{size}(v_{i,j'})}$

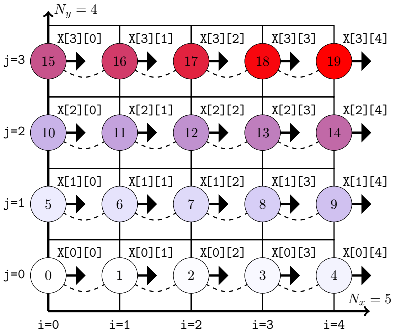
PDE on 2D Cartesian mesh

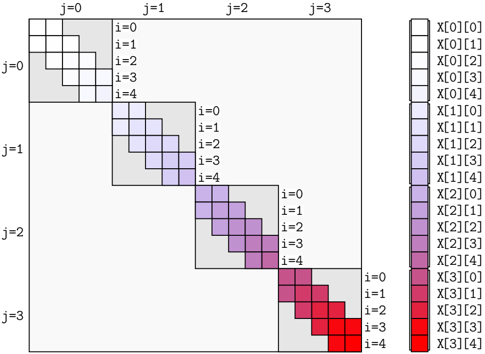
$$-X_{i-1,j} + 2\,X_{i,j} - X_{i+1,j} = h\,B_{i,j}$$



Two-level block matrix structure

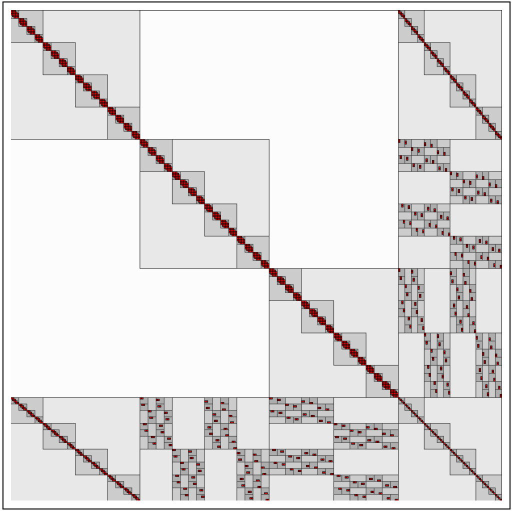
`Diagonal<Tridiagonal>`





LEGOLAS++ template library

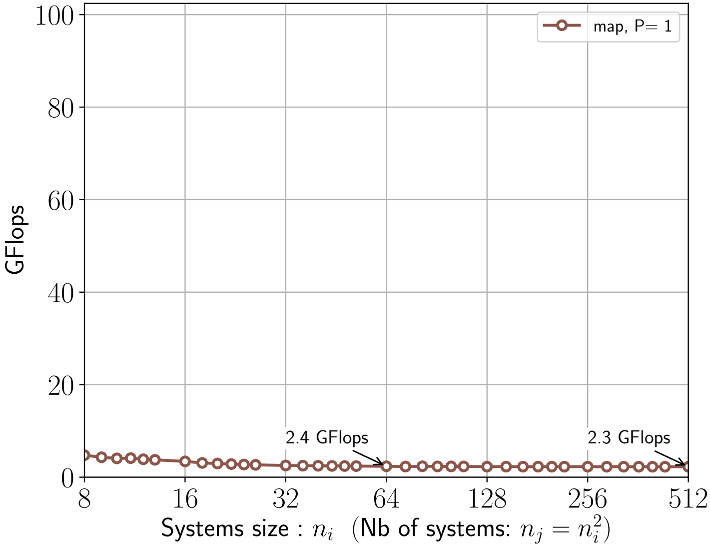
LEGOLAS++ : building blocks for linear algebra solvers.
Highly structured sparse linear algebra.



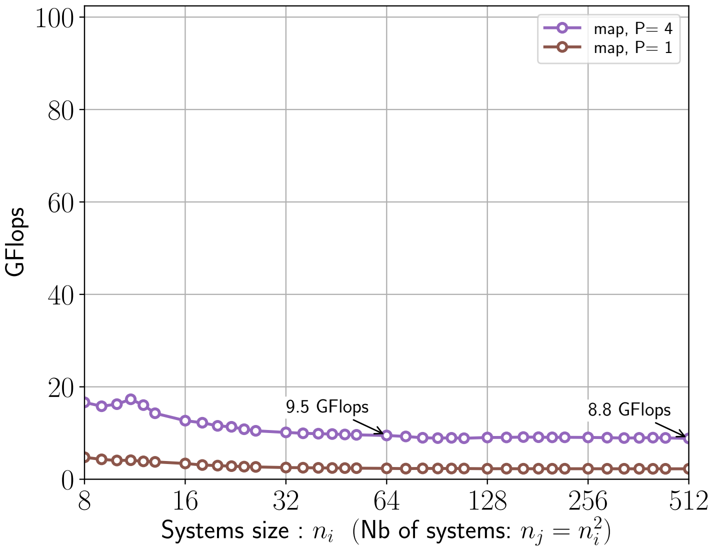
LEGOLAS++ for N tridiagonal systems

```
struct MultiThomasSolver{
  template <class A2D>
  void operator()(int begin, int end,
  A2D D2D, A2D U2D, A2D L2D, A2D B2D, A2D X2D) const{
    typedef typename A2D::Element Element;
    typedef typename A2D::Scalar  Scalar;
    Element S(X2D[0].shape());
    Scalar  one(1.0),s,sm1;
    //Loop over the tridiag systems
    for (int j=begin ; j<end ; j++){
      auto D=D2D[j];auto U=U2D[j]; auto L=L2D[j];
      auto B=B2D[j]; auto X=X2D[j];
      //Thomas Algorithm starts here
      s=D[0]; sm1=one/s;
      const int size=X.size();
      //forward
      X[0]=B[0]*sm1;
      for(int i=1; i<size; i++){
        S[i]=U[i-1]*sm1;
        s=D[i]-L[i]*S[i];
        X[i]=B[i]-L[i]*X[i-1];
        sm1=one/s;
        X[i]*=sm1;
      }
      //backward
      for (int i=(size-2);i>=0 ; i--){
        X[i]-=S[i+1]*X[i+1];
}}}};
```

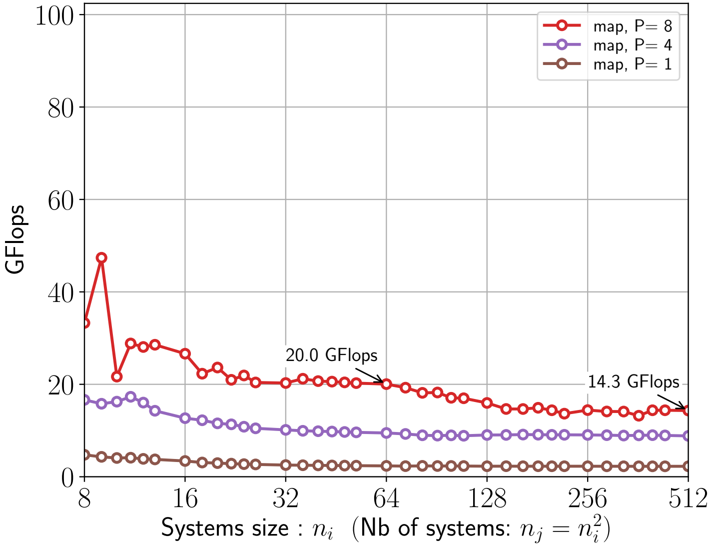
LEGOLAS++ Thomas LDL (Skylake 4-core,4GHz,AVX2)



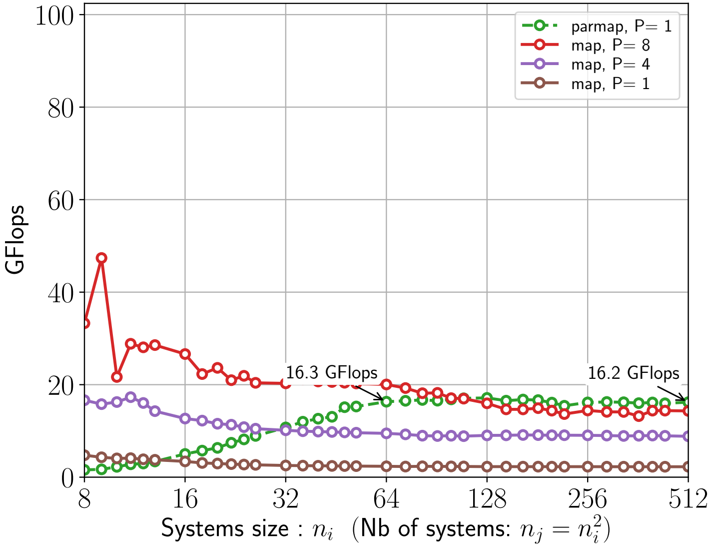
LEGOLAS++ Thomas LDL (Skylake 4-core,4GHz,AVX2)



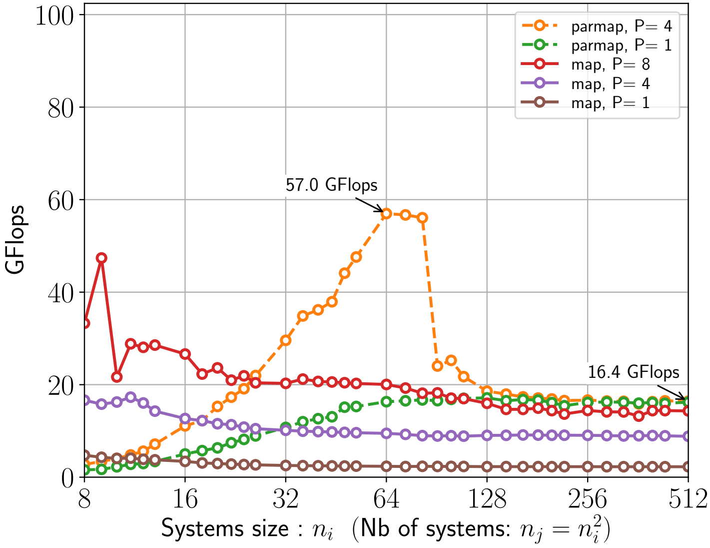
Legolas++ Thomas LDL (Skylake 4-core,4GHz,AVX2)



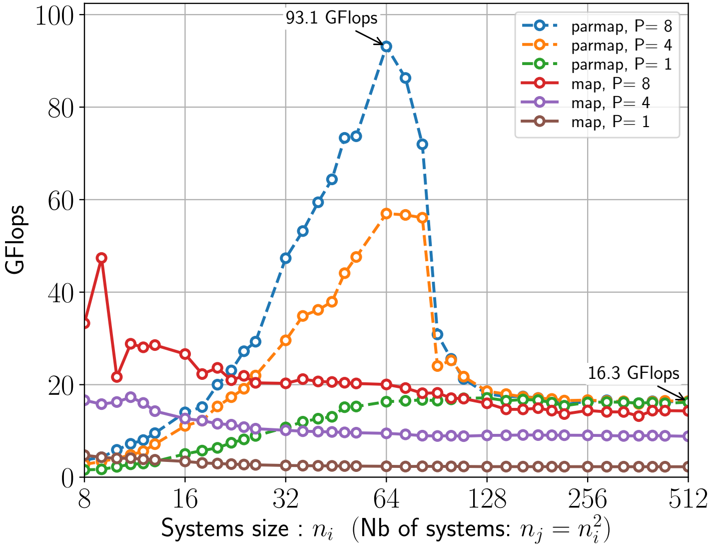
Legolas++ Thomas LDL (Skylake 4-core,4GHz,AVX2)



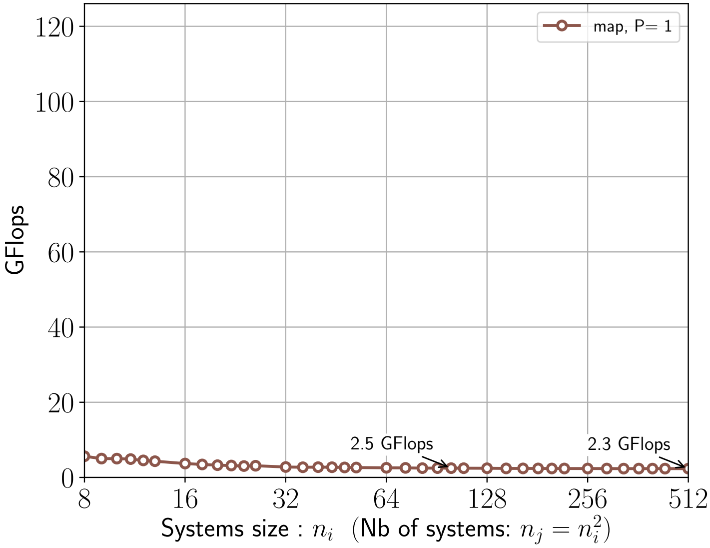
LEGOLAS++ Thomas LDL (Skylake 4-core,4GHz,AVX2)



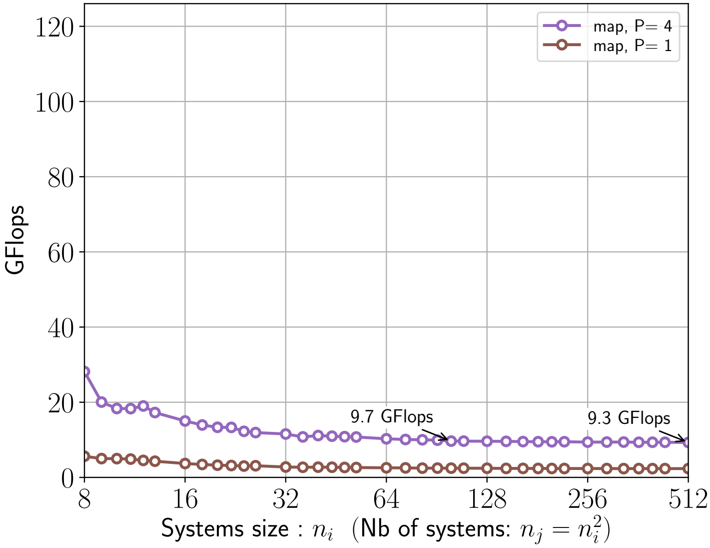
Legolas++ Thomas LDL (Skylake 4-core,4GHz,AVX2)



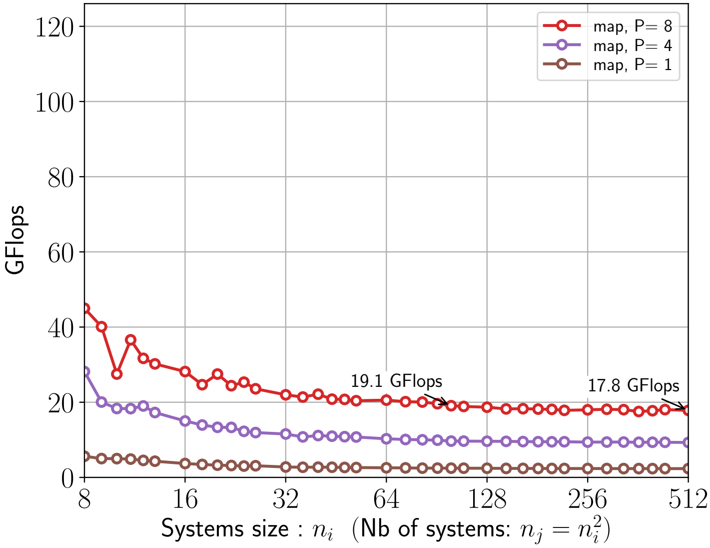
Legolas++ Laplacian (Skylake 4-core,4GHz,AVX2)



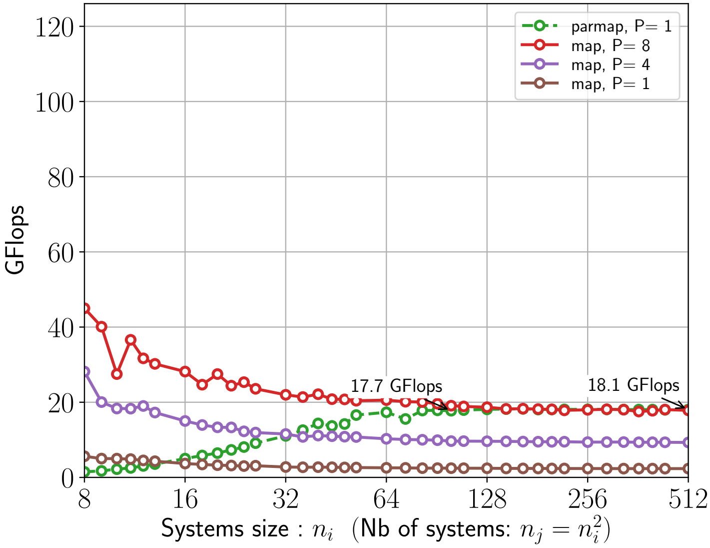
Legolas++ Laplacian (Skylake 4-core,4GHz,AVX2)



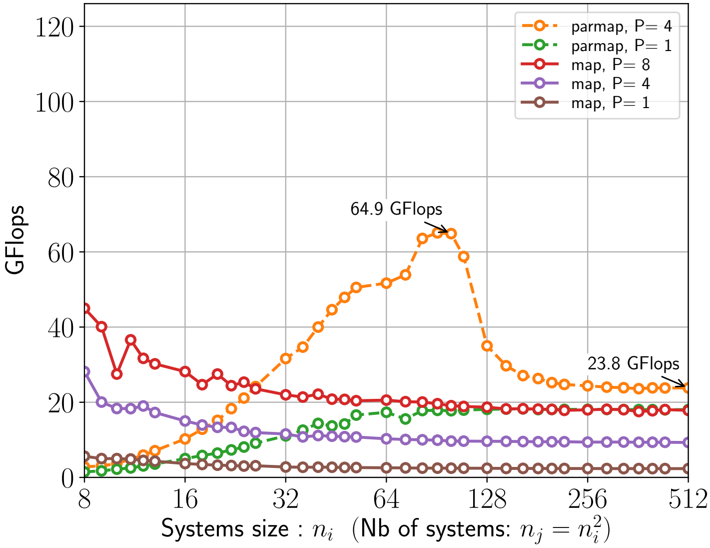
LEGOLAS++ Laplacian (Skylake 4-core,4GHz,AVX2)



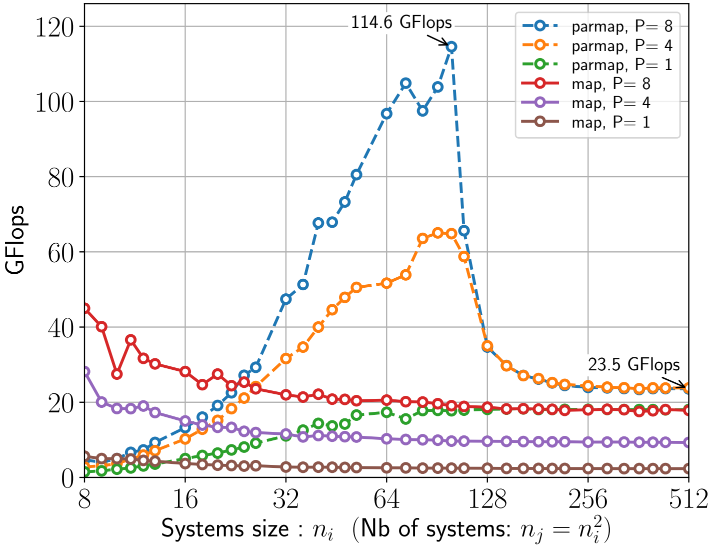
LEGOLAS++ Laplacian (Skylake 4-core,4GHz,AVX2)



Legolas++ Laplacian (Skylake 4-core,4GHz,AVX2)



Legolas++ Laplacian (Skylake 4-core,4GHz,AVX2)



Legolas++ `Array<T,D>` Basic API

`Array<T,D>` Nested Types	
`Scalar`	`Array<T,D>::Scalar` $\hat{=}$ `T`
`Element`	`Array<T,D>::Element` $\hat{=}$ `Array<T,D-1>` if $D > 1$ $\hat{=}$ `T&` if $D = 1$
`Shape`	`int` tuple for array sizes

`Array<T,D>` Methods	
`Array(int n1,..,nD)`	Ctor (variadic)
`Element operator[]`	Element accessor
`int size()`	returns the number of elements
`Shape shape()`	returns `(n1,..,nD)`

A2D= Array<float,2>

```
struct MultiThomasSolver{
  template <class A2D>
  void operator()(int begin, int end,
                  A2D D2D, A2D U2D, A2D L2D
                  A2D B2D, A2D X2D) const{
    typedef typename A2D::Element Element;
    typedef typename A2D::Scalar  Scalar;
    Element S(X2D[0].shape());
    Scalar  one(1.0),s,sm1;
    //Loop on tridiagonal systems
    for (int j=begin ; j<end ; j++){
      auto D=D2D[j];auto U=U2D[j]; auto L=L2D[j];
      auto B=B2D[j]; auto X=X2D[j];
      const int size=X.size();
      X[0]=B[0]*sm1;
      for(int i=1; i<size; i++){
        S[i]=U[i-1]*sm1;
        ...
```

A2D= Array<float,2>

```
struct MultiThomasSolver{
  template <class A2D>
  void operator()(int begin, int end,
                  A2D D2D, A2D U2D, A2D L2D
                  A2D B2D, A2D X2D) const{
    typedef typename A2D::Element Element;//Array<float,1>
    typedef typename A2D::Scalar  Scalar;
    Element S(X2D[0].shape());
    Scalar  one(1.0),s,sm1;
    //Loop on tridiagonal systems
    for (int j=begin ; j<end ; j++){
      auto D=D2D[j];auto U=U2D[j]; auto L=L2D[j];
      auto B=B2D[j]; auto X=X2D[j];
      const int size=X.size();
      X[0]=B[0]*sm1;
      for(int i=1; i<size; i++){
        S[i]=U[i-1]*sm1;
        ...
```

A2D= Array<float,2>

```
struct MultiThomasSolver{
  template <class A2D>
  void operator()(int begin, int end,
                  A2D D2D, A2D U2D, A2D L2D
                  A2D B2D, A2D X2D) const{
    typedef typename A2D::Element Element;//Array<float,1>
    typedef typename A2D::Scalar  Scalar; //float
    Element S(X2D[0].shape());
    Scalar  one(1.0),s,sm1;
    //Loop on tridiagonal systems
    for (int j=begin ; j<end ; j++){
      auto D=D2D[j];auto U=U2D[j]; auto L=L2D[j];
      auto B=B2D[j]; auto X=X2D[j];
      const int size=X.size();
      X[0]=B[0]*sm1;
      for(int i=1; i<size; i++){
        S[i]=U[i-1]*sm1;
        ...
```

Legolas++ for *N* tridiagonal systems

```
int main () {

  size_t ni=200;//System size
  size_t nj=800;//Number of systems

  typedef Legolas::Array<float,2> A2D;

  A2D u(nj,ni),l(nj,ni),d(nj,ni);
  A2D X(nj,ni),B(nj,ni);
  //.. Arrays initialization
  ...
  auto multiThomasSolver=MultiThomasSolver();
  //Solve all systems (sequentially)
  multiThomasSolver(0,nj,d,u,l,B,X);


}
```

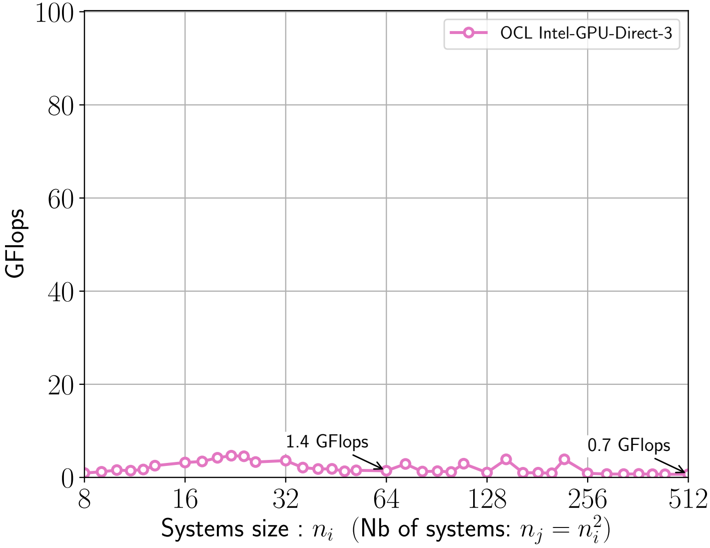
LEGOLAS++ Parallel Expression Template

```
//3D Array containing 100 double elts
Legolas::Array<double,3> X(10,5,2);
Legolas::Array<double,3> Y(10,5,2);
Legolas::Array<double,3> Z(10,5,2);


 for (int k=0; k<10; k++)
  for (int j=0; j<5; j++)
   for (int i=0; j<2; i++)
   Y[k][j][i]+=2.*X[k][j][i]+Z[k][j][i];

Y+=2.*X+Z; //Expression Template
Y+=2.*X+Z || sec; // Sequential
Y+=2.*X+Z || par; // MultiThreaded
```

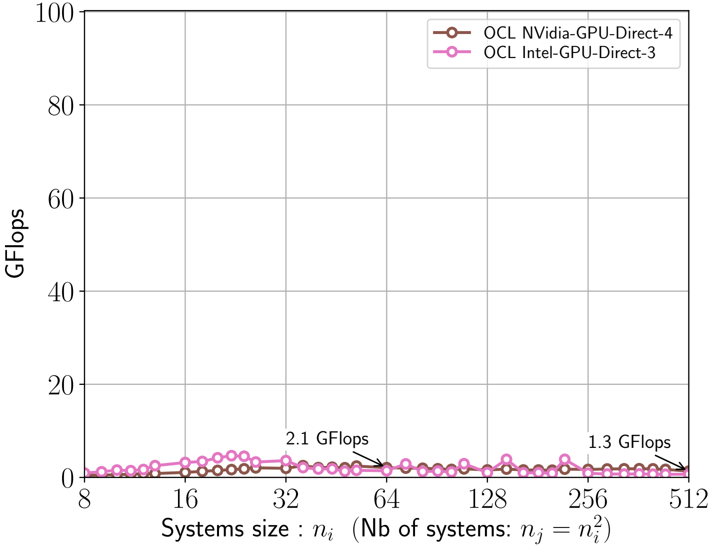
LEGOLAS++ OpenCL Thomas (Direct)

```
__kernel void thomasAlgorithm(
#define SYSTEM_SIZE %(system_size)d
__global float *L,__global float *D,
__global float *U,__global float *X,
__global float *B){
//o: system offset
int o=get_global_id(0)*SYSTEM_SIZE;

float s=D[o+0];
float sm1=1.0/s;
float Sl[SYSTEM_SIZE];
//forward
X[o+0]=B[o+0]*sm1;
for (int i=1 ; i<SYSTEM_SIZE ; i++){
    Sl[i]=U[o+i-1]*sm1;
    s=D[o+i]-L[o+i]*Sl[i];
    X[o+i]=B[o+i]-L[o+i]*X[o+i-1];
    sm1=1./s;
    X[o+i]*=sm1;
}
//backward
for (int i=(SYSTEM_SIZE-2);i>=0 ; i--){
    X[o+i]-=Sl[i+1]*X[o+i+1];
}}
```

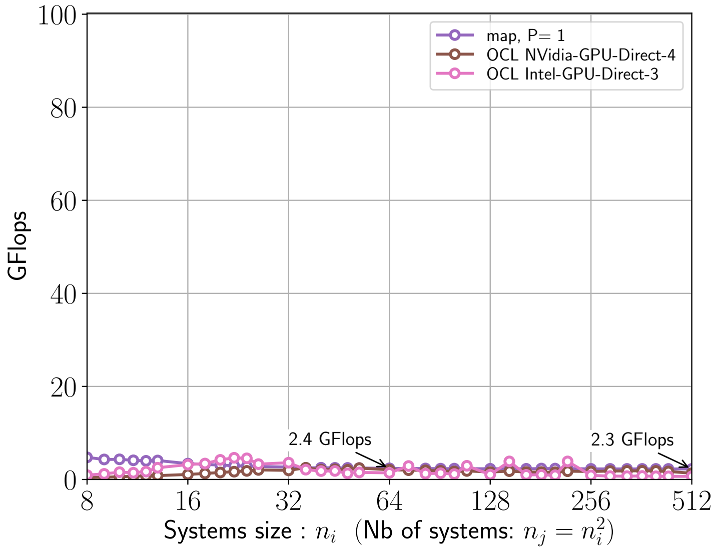
LEGOLAS++ Thomas OpenCL Direct (Skylake)



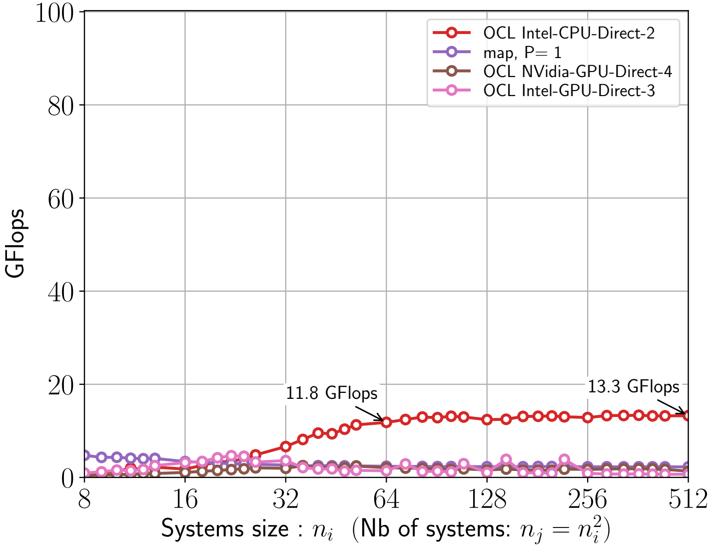
Legolas++ Thomas OpenCL Direct (Skylake)



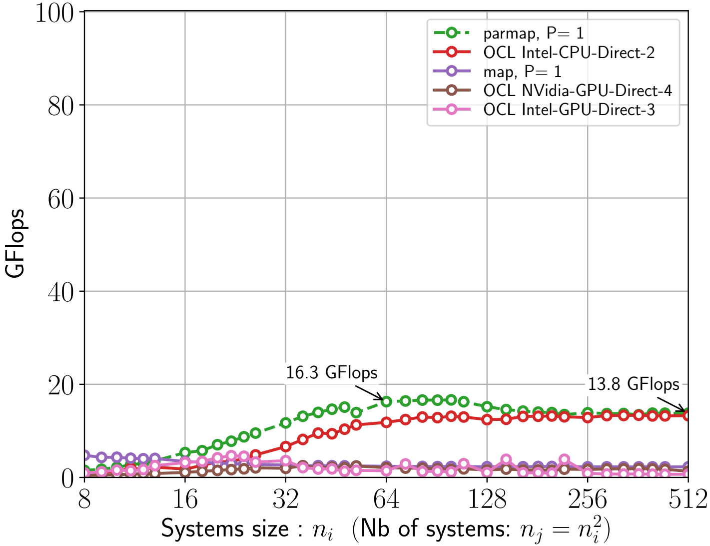
LEGOLAS++ Thomas OpenCL Direct (Skylake)



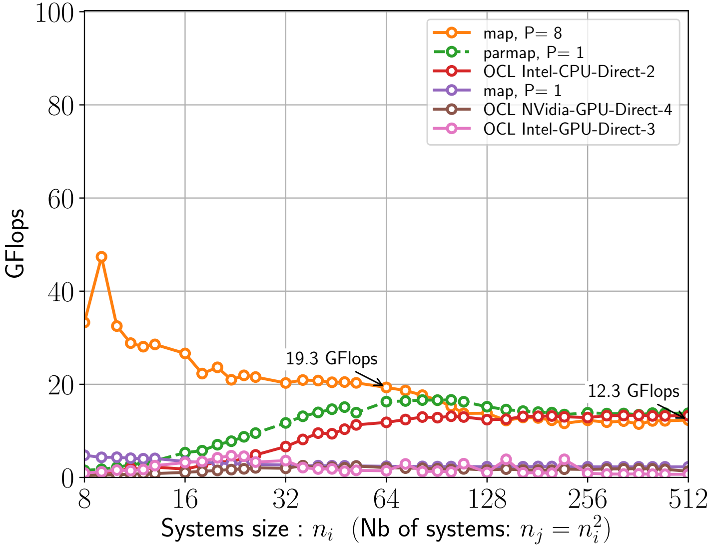
LEGOLAS++ Thomas OpenCL Direct (Skylake)



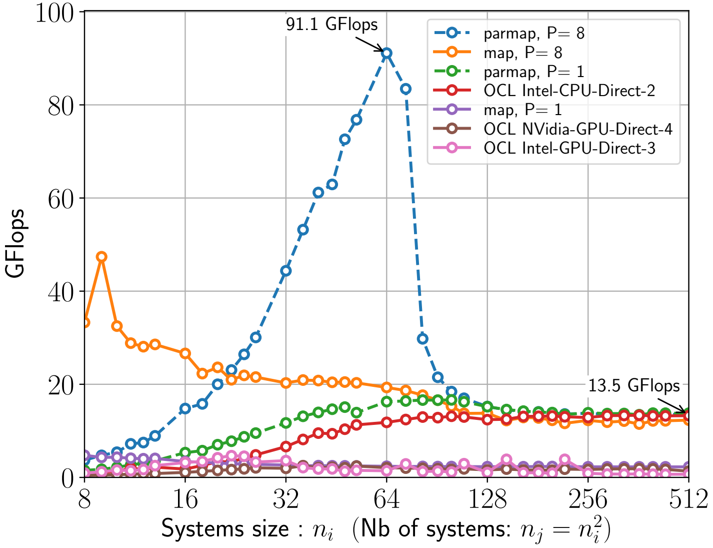
LEGOLAS++ Thomas OpenCL Direct (Skylake)



LEGOLAS++ Thomas OpenCL Direct (Skylake)



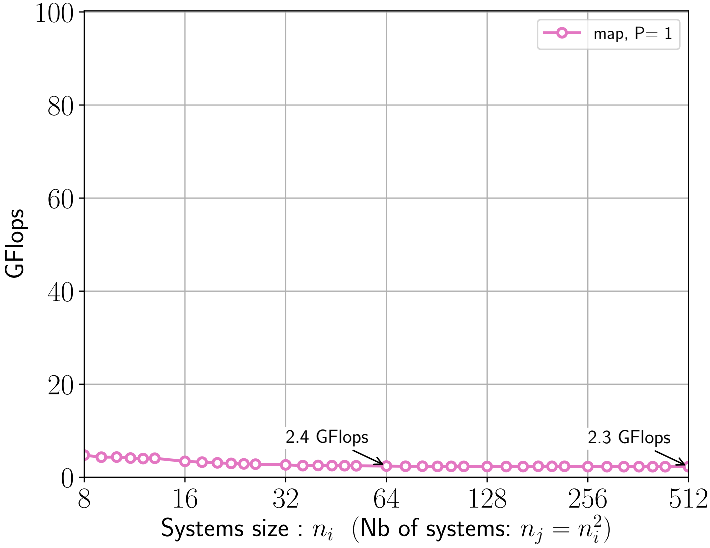
Legolas++ Thomas OpenCL Direct (Skylake)



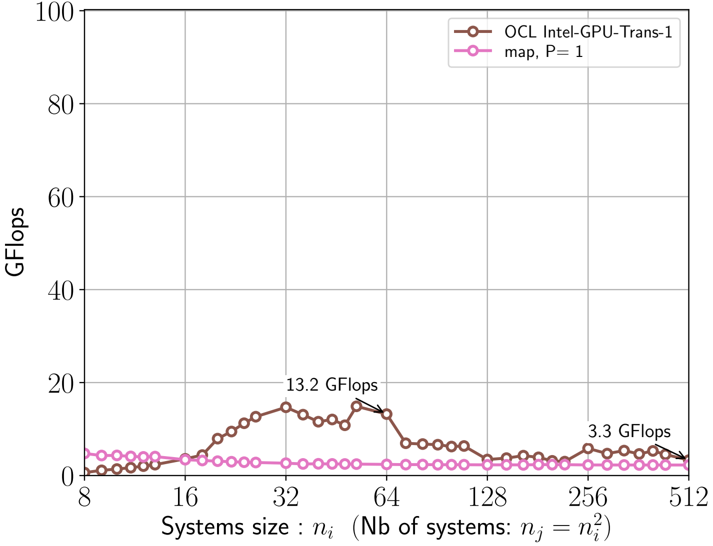
Legolas++ OpenCL Thomas (Transposed)

```
__kernel void thomasAlgorithm(
#define SYSTEM_SIZE %(system_size)d
__global float *L,__global float *D,__global ↩
   float *U,
__global float *X,__global float *B, unsigned ↩
   n_systems)
{
//o: system offset
int gid=get_global_id(0);

float s=D[gid];
float sm1=1.0/s;
//forward
X[gid]=B[gid]*sm1;
//local workspace
float Sl[SYSTEM_SIZE];
for (int i=1 ; i<SYSTEM_SIZE ; i++){
  int I=gid+n_systems*i;
  int Im1=gid+n_systems*(i-1);
  Sl[i]=U[Im1]*sm1;
  s=D[I]-L[I]*Sl[i];
  X[I]=B[I]-L[I]*X[Im1];
  sm1=1./s;
  X[I]*=sm1;
}
//backward
for (int i=(SYSTEM_SIZE-2);i>=0 ; i--){
  int I=gid+n_systems*i;
  int Ip1=gid+n_systems*(i+1);
  X[I]-=Sl[i+1]*X[Ip1];
}}
```

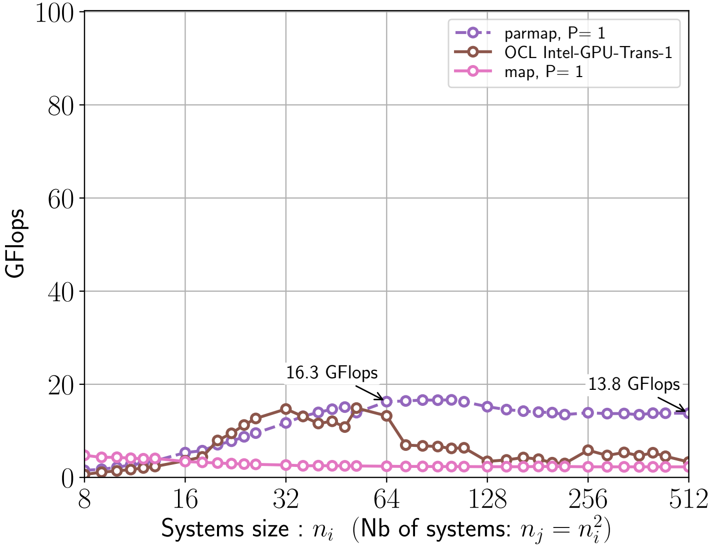
LEGOLAS++ Thomas OpenCL Transposed (Skylake)



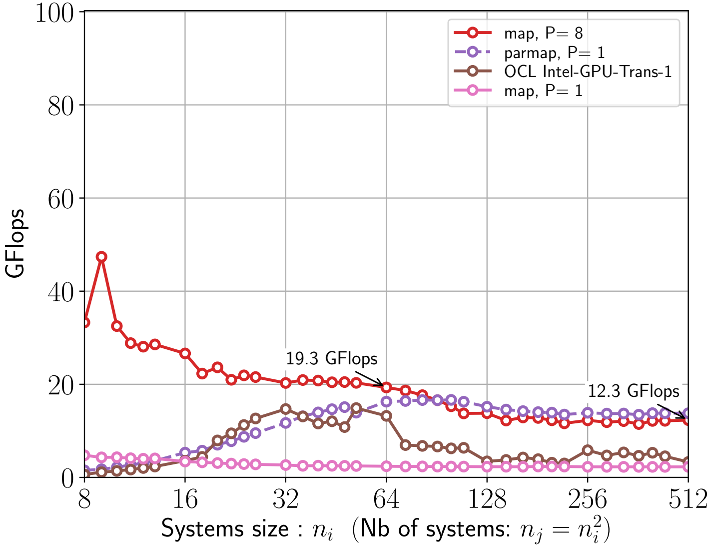
LEGOLAS++ Thomas OpenCL Transposed (Skylake)



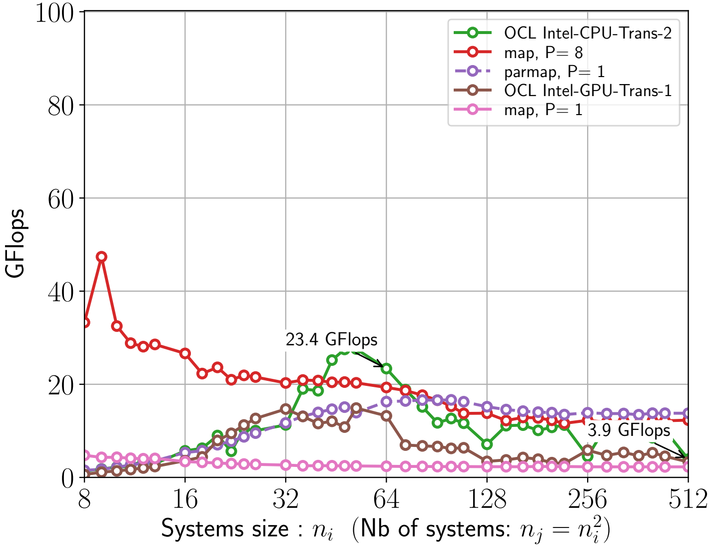
LEGOLAS++ Thomas OpenCL Transposed (Skylake)



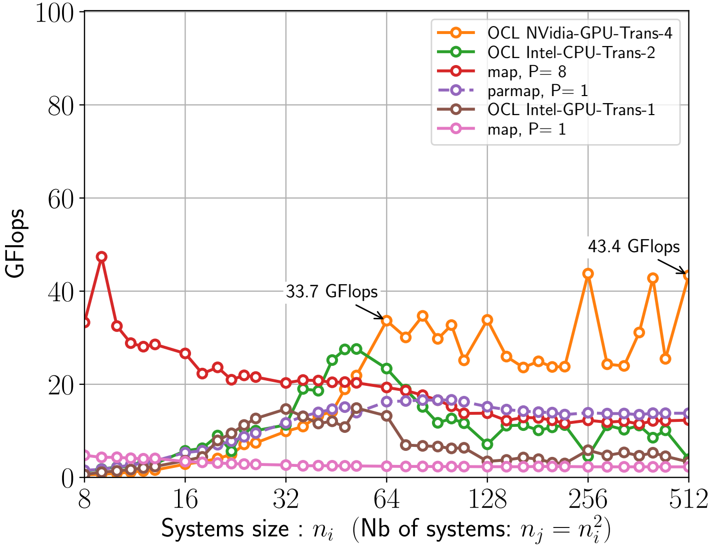
LEGOLAS++ Thomas OpenCL Transposed (Skylake)



LEGOLAS++ Thomas OpenCL Transposed (Skylake)



LEGOLAS++ Thomas OpenCL Transposed (Skylake)



LEGOLAS++ Thomas OpenCL Transposed (Skylake)