An OpenMP source to source compiler for Kaapi and StarPU

Philippe Virouleau

INRIA - MOAIS team

Friday, July 11th

Plan

1. ADT KStar
2. Overview of the compiler
3. Introducing KASTORS benchmark suite
4. Future works

Plan

Context : HPC

Applications

- CPU intensive
- Dense Linear Algebra
- Simulations

Architectures

- Multi-cores/Multi-accelerators
- + Xeon Phi (Many Integrated Core)

Existing runtimes

- StarPU [Runtime]
- XKaapi [MOAIS]
- OMPSS [BSC]

Objectives

Use these architectures

- Existing standard : OpenMP (C/C++/Fortran)
  - OMP 3.1 : tasks and loops
  - OMP 4.0 : dependent tasks, accelerators, SIMD
- + extensions

Execution

- StarPU (http://starpu.gforge.inria.fr)
- XKaapi (http://kaapi.gforge.inria.fr)

OpenMP => Runtime

- Source to source compiler (Clang)
- Support for OpenMP 3.1 + dependencies + accelerators

Who is involved

- Lead
  - T. Gautier [MOAIS] (Grenoble)
- Bordeaux [RUNTIME]
  - O. Aumage (Local lead)
  - N. Furmento
  - S. Thibault
  - ???, IJD
- Grenoble [MOAIS]
  - F. Broquedis
  - P. Brunet, IJD, ADT K'Star
  - P. Virouleau, IJD, ADT Kawah

Plan

Description

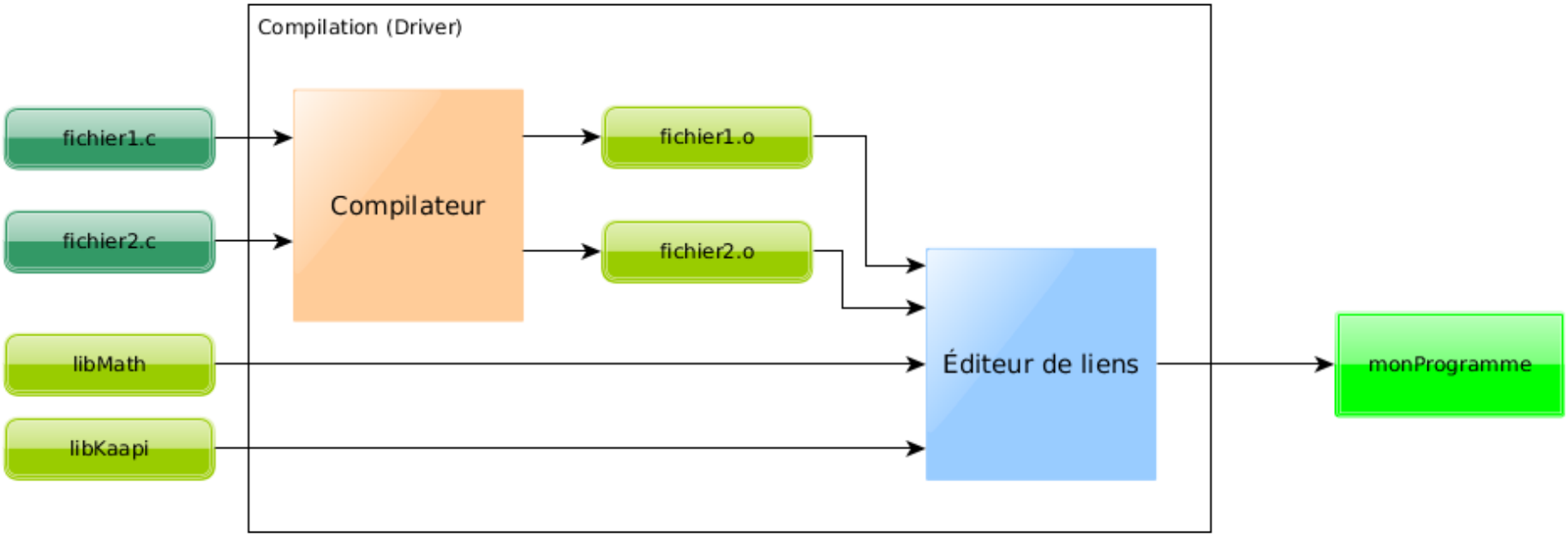
Flow

- Input : C/C++ code with OpenMP.
- Intermediary file : C/C++ code with call to runtime
- Output : program for target platform

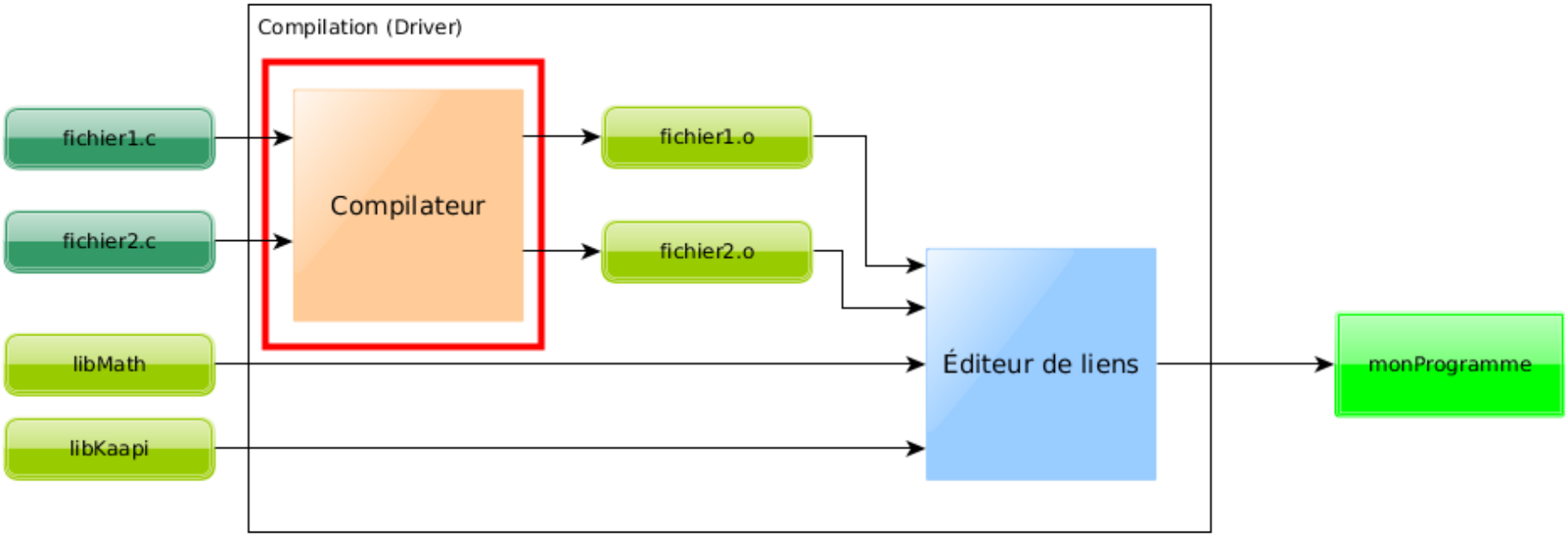
Tools

- Code transformation : Clang
- Final program : user-defined backend compiler (gcc/clang/icc tested)

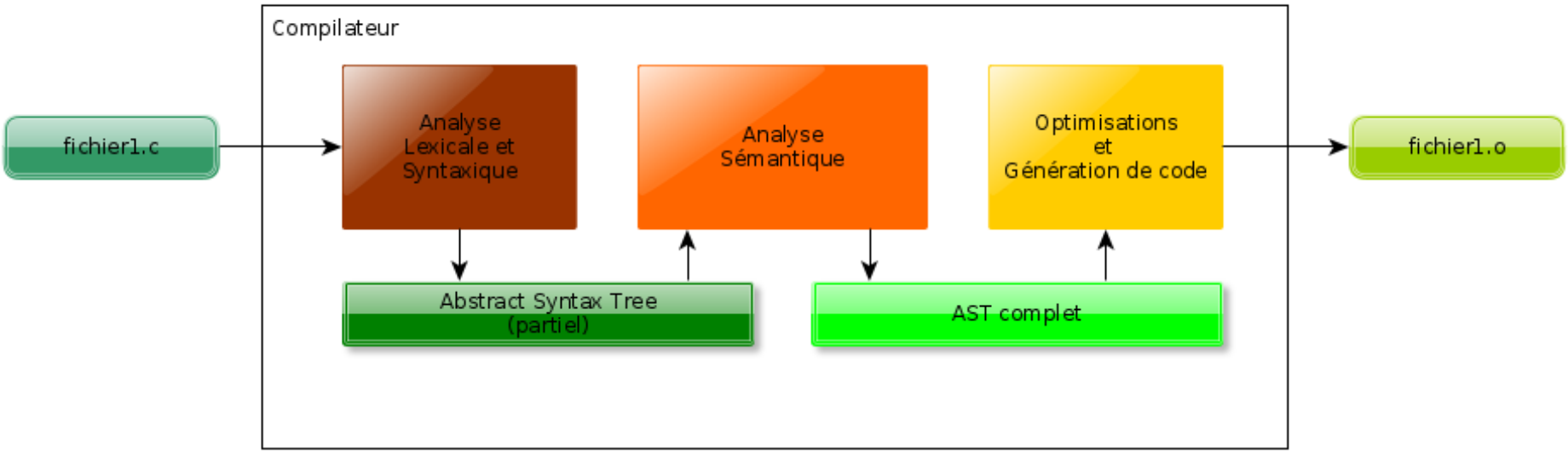
The driver


Compilation (Driver)
fichier1.c
fichier2.c
libMath
libKaapi
Compilateur
fichier1.o
fichier2.o
Éditeur de liens
monProgramme

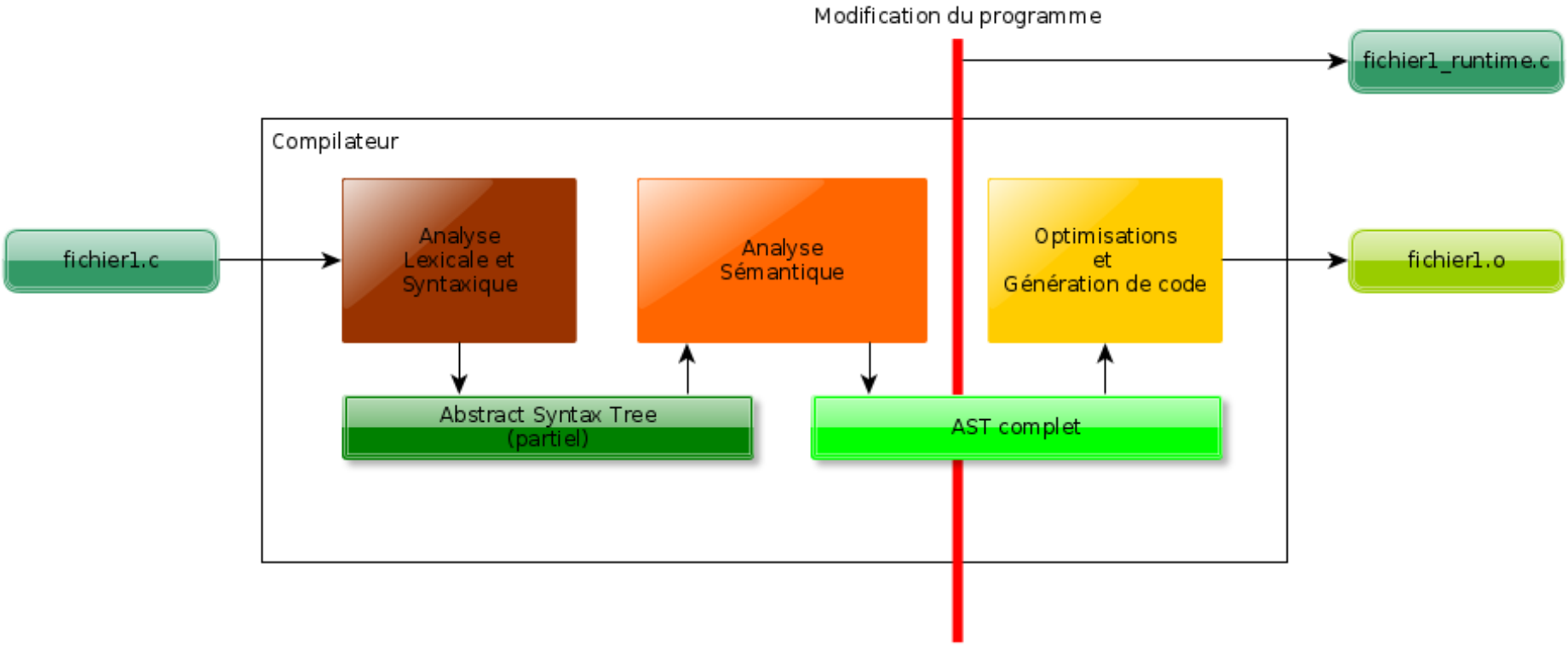
The driver


Compilation (Driver)
fichier1.c
fichier2.c
Compilateur
fichier1.o
fichier2.o
libMath
libKaapi
Éditeur de liens
monProgramme

Within the compiler



Within the compiler



A small example

```
int main()
{
    int x = 3;
    x += 4;
#pragma omp barrier
    return 0;
}
```

A small example

```
int main()
{
    int x = 3;
    x += 4;
#pragma omp barrier
    return 0;
}
```

```
TranslationUnitDecl 0x2ff5610 <<invalid sloc>>
|-TypedefDecl 0x2ff5af0 <<invalid sloc>> __int128_t '__int128'
|-TypedefDecl 0x2ff5b50 <<invalid sloc>> __uint128_t 'unsigned __int128'
|-TypedefDecl 0x2ff5ea0 <<invalid sloc>> __builtin_va_list '__va_list_tag [1]'
`-FunctionDecl 0x2ff5f40 <example.c:2:1, line:8:1> main 'int ()'
  `-CompoundStmt 0x2ff6168 <line:3:1, line:8:1>
    |-DeclStmt 0x2ff6068 <line:4:5, col:14>
    | `-VarDecl 0x2ff5ff0 <col:5, col:13> x 'int'
    |   `-IntegerLiteral 0x2ff6048 <col:13> 'int' 3
    |-CompoundAssignOperator 0x2ff60c8 <line:5:5, col:10> 'int' '+=' ComputeLHSTy='
        int' ComputeResultTy='int'
    | |-DeclRefExpr 0x2ff6080 <col:5> 'int' lvalue Var 0x2ff5ff0 'x' 'int'
    | `-IntegerLiteral 0x2ff60a8 <col:10> 'int' 4
    |-OMPBarrierDirective 0x2ff6100 <line:6:9, col:20>
    `-ReturnStmt 0x2ff6148 <line:7:5, col:12>
      `-IntegerLiteral 0x2ff6128 <col:12> 'int' 0
```

A small example

```
int main()
{
    int x = 3;
    x += 4;
#pragma omp barrier
    return 0;
}
```

```
TranslationUnitDecl 0x2ff5610 <<invalid sloc>>
|-TypedefDecl 0x2ff5af0 <<invalid sloc>> __int128_t '__int128'
|-TypedefDecl 0x2ff5b50 <<invalid sloc>> __uint128_t 'unsigned __int128'
|-TypedefDecl 0x2ff5ea0 <<invalid sloc>> __builtin_va_list '__va_list_tag [1]'
`-FunctionDecl 0x2ff5f40 <example.c:2:1, line:8:1> main 'int ()'
  `-CompoundStmt 0x2ff6168 <line:3:1, line:8:1>
    |-DeclStmt 0x2ff6068 <line:4:5, col:14>
    | `-VarDecl 0x2ff5ff0 <col:5, col:13> x 'int'
    |   `-IntegerLiteral 0x2ff6048 <col:13> 'int' 3
    |-CompoundAssignOperator 0x2ff60c8 <line:5:5, col:10> 'int' '+=' ComputeLHSTy='
        int' ComputeResultTy='int'
    | |-DeclRefExpr 0x2ff6080 <col:5> 'int' lvalue Var 0x2ff5ff0 'x' 'int'
    | `-IntegerLiteral 0x2ff60a8 <col:10> 'int' 4
    |-OMPBarrierDirective 0x2ff6100 <line:6:9, col:20>
    `-ReturnStmt 0x2ff6148 <line:7:5, col:12>
      `-IntegerLiteral 0x2ff6128 <col:12> 'int' 0
```

Visiting the AST

```
class RuntimeVisitor : public clang::RecursiveASTVisitor<RuntimeVisitor>
{
    public:
    /*...*/
        bool VisitOMPTaskDirective(clang::OMPTaskDirective *s);
        bool VisitOMPTaskwaitDirective(clang::OMPTaskwaitDirective *s);
        bool VisitFunctionDecl(clang::FunctionDecl *f);
    /*...*/
};
```

What's next ?

Modify the AST ?

"Clang is not designed to support mutation of its AST" Richard Smith.

What's next ?

Modify the AST ?

"Clang is not designed to support mutation of its AST" Richard Smith.

Use the Rewriter

- Many ways to visit the AST ...
- With some constraints.
- Does the job !

Visiting the AST

```
class RuntimeVisitor : public clang::RecursiveASTVisitor<RuntimeVisitor>
{
    public:
    /*...*/
        bool VisitOMPTaskDirective(clang::OMPTaskDirective *s);
        bool VisitOMPTaskwaitDirective(clang::OMPTaskwaitDirective *s);
        bool VisitFunctionDecl(clang::FunctionDecl *f);
    /*...*/
};
```

Visiting the AST

```
class RuntimeVisitor : public clang::RecursiveASTVisitor<RuntimeVisitor>
{
    public:
    /*...*/
        bool VisitOMPTaskDirective(clang::OMPTaskDirective *s);
        bool VisitOMPTaskwaitDirective(clang::OMPTaskwaitDirective *s);
        bool VisitFunctionDecl(clang::FunctionDecl *f);
    /*...*/
};

class Rewriter {
    /*...*/
    bool InsertTextAfter(SourceLocation Loc, StringRef Str);
    bool InsertTextBefore(SourceLocation Loc, StringRef Str);
    /*...*/
    bool RemoveText(SourceLocation Start, unsigned Length,
                    RewriteOptions opts = RewriteOptions());
    /*...*/
    bool ReplaceText(SourceRange range, StringRef NewStr);
};
```

Basic rewriting example

```
void print(const char *msg) {
    printf("%s\n", msg);
}

int main( int argc, char** argv) {
    char mon_msg[] = "Hello !";
#pragma omp task
    print(mon_msg);
#pragma omp taskwait
    return 0;
}
```

```
void print(const char *msg) {
    printf("%s\n", msg);
}

/* Generated arg struct */
struct __gen_argstruct {
  char [8] mon_msg;
};

int main( int argc, char** argv) {
    char mon_msg[] = "Hello !";
  /* Generated task spawn */
  omp_push_task(wrapper, args);
  /* Generated taskwait */
  omp_sched_sync();
  return 0;
}

/* Generated wrapper */
void wrapper(void *_k_arg) {
  // The captured stmts param
  __gen_argstruct *args = (__gen_argstruct
        *)_k_arg;
  char [8] mon_msg = args->mon_msg;
  // The captured stmts
  print(mon_msg);
}
/* ... */
```

Basic rewriting example

```
void print(const char *msg) {
    printf("%s\n", msg);
}

int main( int argc, char** argv) {
    char mon_msg[] = "Hello !";
#pragma omp task
    print(mon_msg);
#pragma omp taskwait
    return 0;
}
```

Rewriting steps

- Runtime-specific type creation

```
void print(const char *msg) {
    printf("%s\n", msg);
}

/* Generated arg struct */
struct __gen_argstruct {
  char [8] mon_msg;
};

int main( int argc, char** argv) {
    char mon_msg[] = "Hello !";
  /* Generated task spawn */
  omp_push_task(wrapper, args);
  /* Generated taskwait */
  omp_sched_sync();
  return 0;
}

/* Generated wrapper */
void wrapper(void *_k_arg) {
  // The captured stmts param
  __gen_argstruct *args = (__gen_argstruct
        *)_k_arg;
  char [8] mon_msg = args->mon_msg;
  // The captured stmts
  print(mon_msg);
}
/* ... */
```

Basic rewriting example

```
void print(const char *msg) {
    printf("%s\n", msg);
}

int main( int argc, char** argv) {
    char mon_msg[] = "Hello !";
#pragma omp task
    print(mon_msg);
#pragma omp taskwait
    return 0;
}
```

Rewriting steps

- Runtime-specific type creation
- Code wrapper creation

```
void print(const char *msg) {
    printf("%s\n", msg);
}

/* Generated arg struct */
struct __gen_argstruct {
  char [8] mon_msg;
};

int main( int argc, char** argv) {
    char mon_msg[] = "Hello !";
  /* Generated task spawn */
  omp_push_task(wrapper, args);
  /* Generated taskwait */
  omp_sched_sync();
  return 0;
}

/* Generated wrapper */
void wrapper(void *_k_arg) {
  // The captured stmts param
  __gen_argstruct *args = (__gen_argstruct
        *)_k_arg;
  char [8] mon_msg = args->mon_msg;
  // The captured stmts
  print(mon_msg);
}
/* ... */
```

Basic rewriting example

```
void print(const char *msg) {
    printf("%s\n", msg);
}

int main( int argc, char** argv) {
    char mon_msg[] = "Hello !";
#pragma omp task
    print(mon_msg);
#pragma omp taskwait
    return 0;
}
```

Rewriting steps

- Runtime-specific type creation
- Code wrapper creation
- Task creation in the runtime

```
void print(const char *msg) {
    printf("%s\n", msg);
}

/* Generated arg struct */
struct __gen_argstruct {
  char [8] mon_msg;
};

int main( int argc, char** argv) {
    char mon_msg[] = "Hello !";
  /* Generated task spawn */
  omp_push_task(wrapper, args);
  /* Generated taskwait */
  omp_sched_sync();
  return 0;
}

/* Generated wrapper */
void wrapper(void *_k_arg) {
  // The captured stmts param
  __gen_argstruct *args = (__gen_argstruct
        *)_k_arg;
  char [8] mon_msg = args->mon_msg;
  // The captured stmts
  print(mon_msg);
}
/* ... */
```

Basic rewriting example

```
void print(const char *msg) {
    printf("%s\n", msg);
}

int main( int argc, char** argv) {
    char mon_msg[] = "Hello !";
#pragma omp task
    print(mon_msg);
#pragma omp taskwait
    return 0;
}
```

Rewriting steps

- Runtime-specific type creation
- Code wrapper creation
- Task creation in the runtime

```
void print(const char *msg) {
    printf("%s\n", msg);
}

/* Generated arg struct */
struct __gen_argstruct {
  char [8] mon_msg;
};

int main( int argc, char** argv) {
    char mon_msg[] = "Hello !";
  /* Generated task spawn */
  omp_push_task(wrapper, args);
  /* Generated taskwait */
  omp_sched_sync();
  return 0;
}

/* Generated wrapper */
void wrapper(void *_k_arg) {
  // The captured stmts param
  __gen_argstruct *args = (__gen_argstruct
        *)_k_arg;
  char [8] mon_msg = args->mon_msg;
  // The captured stmts
  print(mon_msg);
}
/* ... */
```

Code outlining and data sharing

```
int main (int argc, char *argv[])
{
int i, n;
float a[100], b[100], sum;

#pragma omp parallel for reduction(+:sum) firstprivate(n) shared(sum, a, b)
    for (i=0; i < n; i++) {
        int toto = 5;
        sum = sum + (a[i] * b[i]);
    }
}
```

```
[...]
|   `-OMPParallelDirective 0x2e3c638 <line:6:9, col:42>
|     `-CapturedStmt 0x2e3c598 <col:9, col:42>
|       |-Capture byref Var 0x2df41a0 'i' 'int'
|       |-Capture byref Var 0x2df4210 'n' 'int'
|       |-Capture byref Var 0x2e388f0 'sum' 'float'
|       |-Capture byref Var 0x2e387e0 'a' 'float [100]'
|       |-Capture byref Var 0x2e38880 'b' 'float [100]'
[...]
```

Summary

Steps

- Build an AST through Clang
- Visit and extract informations from it
- Generate correct calls to runtime (XKaapi + StarPU)
- Output final code

Support

Support for OpenMP 3.1 + dependent tasks and SIMD

On the web

http://kstar.gforge.inria.fr

Plan

Motivations

Existing benchmark suite

- BOTS (Barcelona OpenMP Task Suite [BSC])
- NAS, SPECOMP, ...
- No benchmark for dependent tasks yet !

Objectives

- Evaluate the OpenMP dependent task paradigm
- Show OpenMP 4.0 limits for dependent task
- Compare runtimes

OpenMP Dependent Task Suite

Applications

- Jacobi/Poisson2D
- SparseLU (BOTS[a])
- Strassen (BOTS)
- Plasma (2 kernels)

[a] Barcelona OpenMP Task Suite

OpenMP Dependent Task Suite

Jacobi/Poisson2D

Solve poisson equation on a [0,1]x[0,1] square, divided in NxN points.

OpenMP Dependent Task Suite

Jacobi/Poisson2D

Solve poisson equation on a [0,1]x[0,1] square, divided in NxN points.

SparseLU (BOTS)

LU factorization of a sparse matrix

OpenMP Dependent Task Suite

Jacobi/Poisson2D

Solve poisson equation on a [0,1]x[0,1] square, divided in NxN points.

SparseLU (BOTS)

LU factorization of a sparse matrix

Strassen (BOTS)

Dense matrix multiplication by bloc decomposition

OpenMP Dependent Task Suite

Jacobi/Poisson2D

Solve poisson equation on a [0,1]x[0,1] square, divided in NxN points.

SparseLU (BOTS)

LU factorization of a sparse matrix

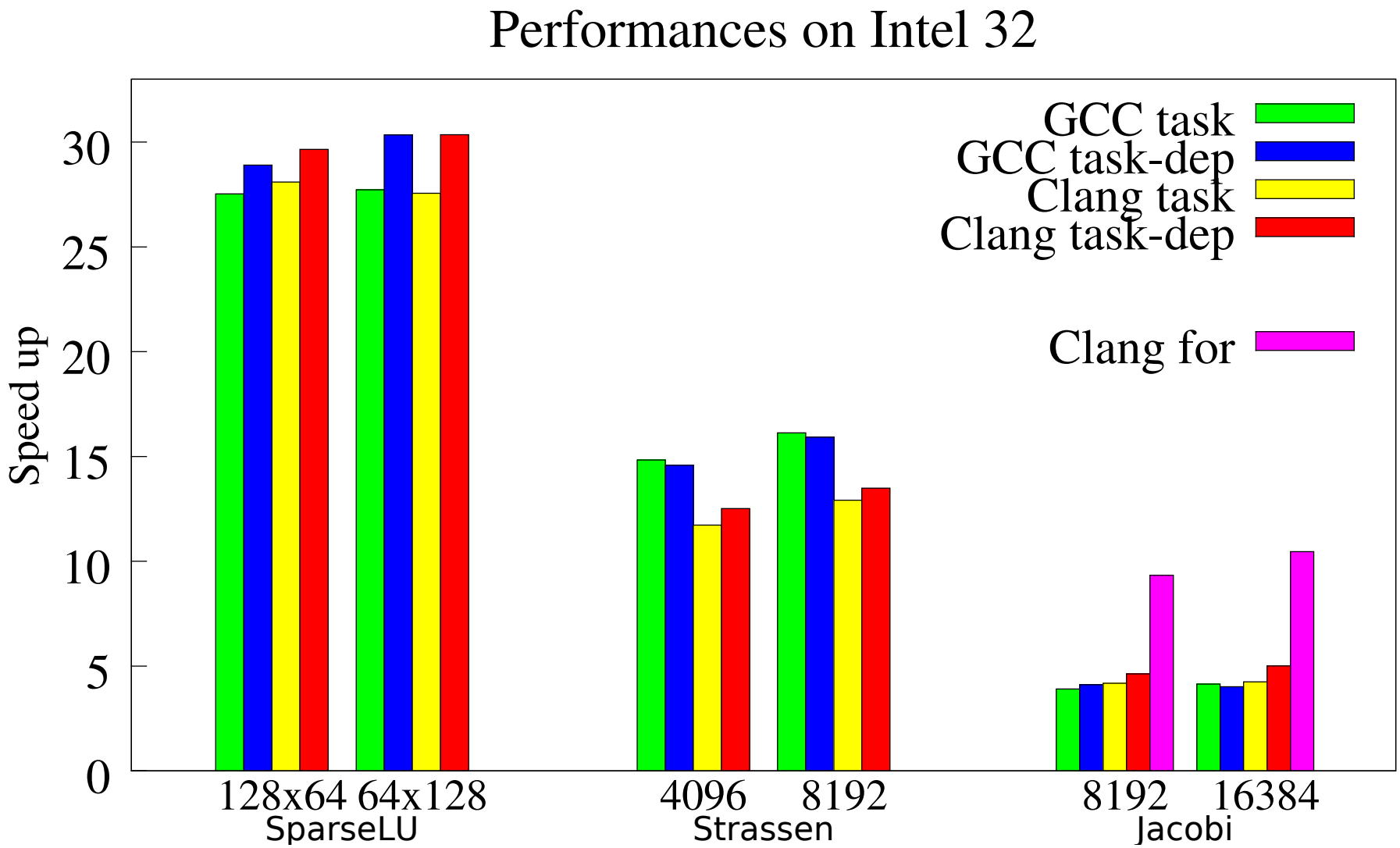
Strassen (BOTS)

Dense matrix multiplication by bloc decomposition

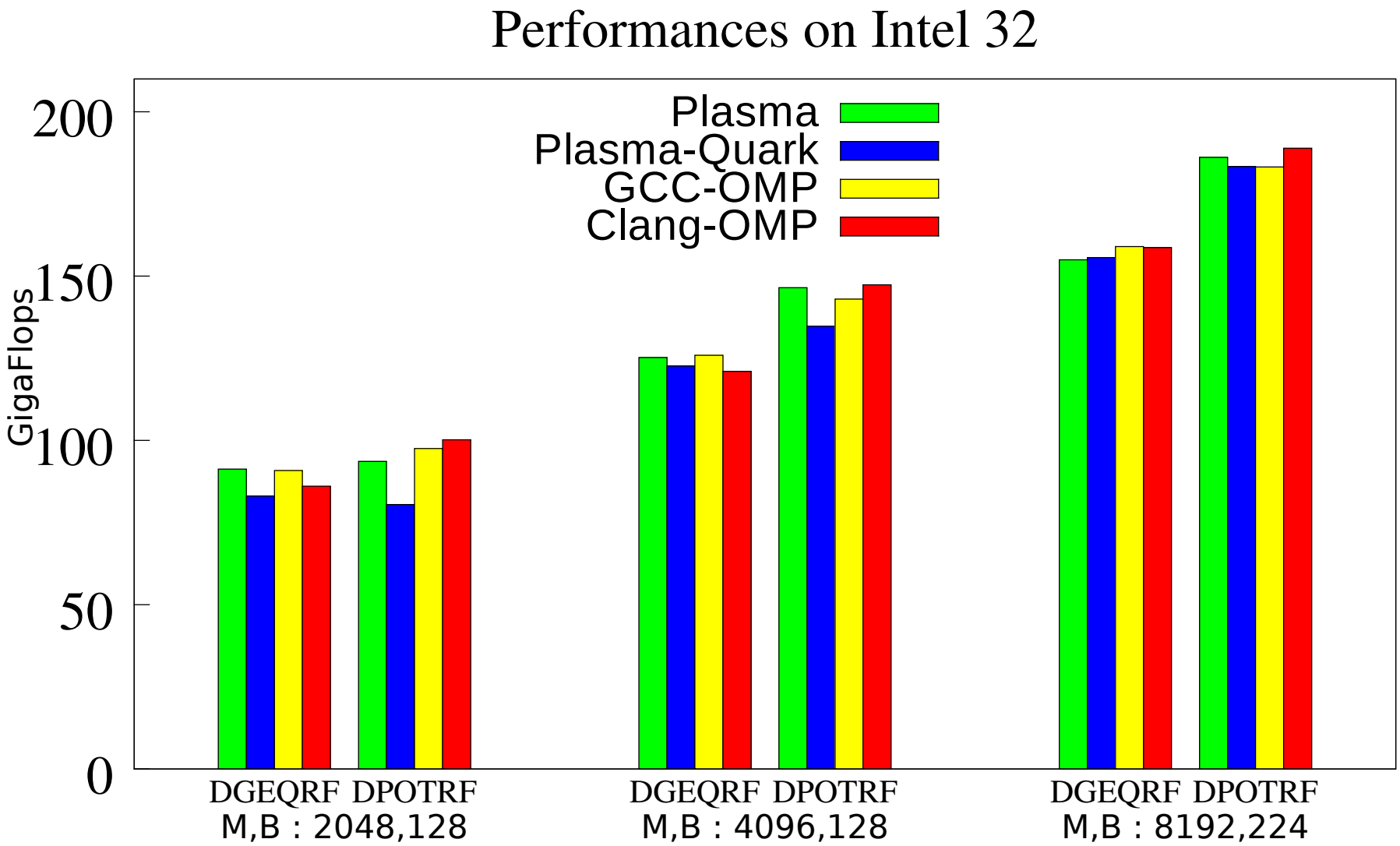
Plasma

- Refactor of the library
- Cholesky and QR factorization on double (dpotrf & dgeqrf)

Performances for SparseLU, Strassen, and Jacobi



Performances for Plasma



Conclusion

Results are quite encouraging

- Better performances with dependent tasks
- Jacobi : scheduling issue (for is better)
- OpenMP 4.0 is a good alternative to QUARK
- Others dedicated runtimes worth a look

On the web

http://kastors.gforge.inria.fr

Plan

Klang improvements

Language

- C++ (support templates)
- Fortran

OpenMP support

- omp target
- leading dimension (avoid VLA)
- reduction in task (Comitee discussion)

KASTORS improvements

Kernels

- Include non Linear Algebra applications
- Improve some performances on Plasma

Papers

- Accepted @ IWOMP2014
- Submission to LLVM/Clang workshop @SC'14 ?

Questions ?