Exloiting multi-level parallelism on Intel KNL
Round Table KNL

Terry Cojean, STORM team

INRIA Bordeaux, LaBRI, Université de Bordeaux

INRIA Bordeaux

March 28\textsuperscript{th}, 2017
Outline

1. The StarPU runtime system
   - Aim of runtime systems
   - The Sequential Task Flow (STF) model

2. Paradigm evolution: Parallel tasks in StarPU

3. Resource aggregation experiments on the Intel KNL
   - Machine and kernels overview
   - Parallel tasks performance
The classical approach is based on a mixture of technologies (e.g., OpenMP).
The classical approach is based on a mixture of technologies (e.g., OpenMP + CUDA)
The classical approach is based on a mixture of technologies (e.g., OpenMP+CUDA+MPI)
The classical approach is based on a mixture of technologies (e.g., OpenMP+CUDA+MPI)
- requires a big programming effort.
- is prone to (performance) portability issues.
The classical approach is based on a mixture of technologies (e.g., OpenMP+CUDA+MPI)
- requires a big programming effort.
- is prone to (performance) portability issues.

**runtimes** provide an abstraction layer that hides the architecture details.

**the workload is expressed as a DAG (Directed Acyclic Graph)** of tasks **scheduled** by the runtime.
Sequential Task Flow (STF) Cholesky algorithm submission

for (j = 0; j < N; j++) {
    submit(POTRF, A[j][j]:RW);
    for (i = j+1; i < N; i++)
        submit(TRSM, A[i][j]:RW, A[j][j]:R);
    for (i = j+1; i < N; i++) {
        submit(SYRK, A[i][i]:RW, A[i][j]:R);
        for (k = j+1; k < i; k++)
            submit(GEMM, A[i][k]:RW,
                   A[i][j]:R, A[k][j]:R);
    }
}
__wait__();
for (j = 0; j < N; j++) {
    submit(POTRF, A[j][j]:RW);
    for (i = j+1; i < N; i++)
        submit(TRSM, A[i][j]:RW, A[j][j]:R);
    for (i = j+1; i < N; i++) {
        submit(SYRK, A[i][i]:RW, A[i][j]:R);
        for (k = j+1; k < i; k++)
            submit(GEMM, A[i][k]:RW,
                   A[i][j]:R, A[k][j]:R);
    }
}__wait__;
Sequential Task Flow (STF) Cholesky algorithm submission

for (j = 0; j < N; j++) {
    submit(POTRF, A[j][j]:RW);
    for (i = j+1; i < N; i++)
        submit(TRSM, A[i][j]:RW, A[j][j]:R);
    for (i = j+1; i < N; i++) {
        submit(SYRK, A[i][i]:RW, A[i][j]:R);
        for (k = j+1; k < i; k++)
            submit(GEMM, A[i][k]:RW, A[i][j]:R,
                    A[i][j]:R, A[k][j]:R);
    }
}__wait__();
for (j = 0; j < N; j++) {
    submit(POTRF, A[j][j]:RW);
    for (i = j+1; i < N; i++)
        submit(TRSM, A[i][j]:RW, A[j][j]:R);
    for (i = j+1; i < N; i++) {
        submit(SYRK, A[i][i]:RW, A[i][j]:R);
        for (k = j+1; k < i; k++)
            submit(GEMM, A[i][k]:RW,
                       A[i][j]:R, A[k][j]:R);
    }
}

__wait__();
Sequential Task Flow (STF) Cholesky algorithm submission

```c
for (j = 0; j < N; j++) {
    submit(POTRF, A[j][j]:RW);
    for (i = j+1; i < N; i++)
        submit(TRSM, A[i][j]:RW, A[j][j]:R);
    for (i = j+1; i < N; i++) {
        submit(SYRK, A[i][i]:RW, A[i][j]:R);
        for (k = j+1; k < i; k++)
            submit(GEMM, A[i][k]:RW,
                   A[i][j]:R, A[k][j]:R);
    }
} __wait__();
```
for (j = 0; j < N; j++) {
    submit(POTRF, A[j][j]:RW);
    for (i = j+1; i < N; i++)
        submit(TRSM, A[i][j]:RW, A[j][j]:R);
    for (i = j+1; i < N; i++) {
        submit(SYRK, A[i][i]:RW, A[i][j]:R);
        for (k = j+1; k < i; k++)
            submit(GEMM, A[i][k]:RW,
                   A[i][j]:R, A[k][j]:R);
    }
}
__wait__();
for (j = 0; j < N; j++) {
    submit(POTRF, A[j][j]:RW);
    for (i = j+1; i < N; i++) {
        submit(TRSM, A[i][j]:RW, A[j][j]:R);
        for (i = j+1; i < N; i++) {
            submit(SYRK, A[i][i]:RW, A[i][j]:R);
            for (k = j+1; k < i; k++)
                submit(GEMM, A[i][k]:RW,
                        A[i][j]:R, A[k][j]:R);
        }
    }
} __wait__;
for (j = 0; j < N; j++) {
    submit(POTRF, A[j][j]:RW);
    for (i = j+1; i < N; i++) {
        submit(TRSM, A[i][j]:RW, A[j][j]:R);
        submit(SYRK, A[i][i]:RW, A[i][j]:R);
        for (k = j+1; k < i; k++)
            submit(GEMM, A[i][k]:RW, A[i][j]:R, A[k][j]:R);
    }
}__wait__();
for (j = 0; j < N; j++) {
    submit(POTRF, A[j][j]:RW);
    for (i = j+1; i < N; i++)
        submit(TRSM, A[i][j]:RW, A[j][j]:R);
    for (i = j+1; i < N; i++) {
        submit(SYRK, A[i][i]:RW, A[i][j]:R);
        for (k = j+1; k < i; k++)
            submit(GEMM, A[i][k]:RW,
                   A[i][j]:R, A[k][j]:R);
    }
}
for (j = 0; j < N; j++) {
    submit(POTRF, A[j][j]:RW);
    for (i = j+1; i < N; i++)
        submit(TRSM, A[i][j]:RW, A[j][j]:R);
    for (i = j+1; i < N; i++) {
        submit(SYRK, A[i][i]:RW, A[i][j]:R);
        for (k = j+1; k < i; k++)
            submit(GEMM, A[i][k]:RW, A[i][j]:R, A[k][j]:R);
    }
}
__wait__;
for (j = 0; j < N; j++) {
    submit(POTRF, A[j][j]:RW);
    for (i = j+1; i < N; i++) {
        submit(TRSM, A[i][j]:RW, A[j][j]:R);
        for (k = j+1; k < i; k++)
            submit(GEMM, A[i][k]:RW,
                    A[i][j]:R, A[k][j]:R);
    }
}
__wait__();
for (j = 0; j < N; j++) {
    submit(POTRF, A[j][j]:RW);
    for (i = j+1; i < N; i++) {
        submit(TRSM, A[i][j]:RW, A[j][j]:R);
        for (i = j+1; i < N; i++) {
            submit(SYRK, A[i][i]:RW, A[i][j]:R);
            for (k = j+1; k < i; k++)
                submit(GEMM, A[i][k]:RW, A[i][j]:R, A[k][j]:R);
        }
    }
    __wait__();
}
for (j = 0; j < N; j++) {
    submit(POTRF, A[j][j]:RW);
    for (i = j+1; i < N; i++)
        submit(TRSM, A[i][j]:RW, A[j][j]:R);
    for (i = j+1; i < N; i++) {
        submit(SYRK, A[i][i]:RW, A[i][j]:R);
        for (k = j+1; k < i; k++)
            submit(GEMM, A[i][k]:RW, A[i][j]:R, A[k][j]:R);
    }
} __wait__();
Tasks execution on a heterogeneous node

CPU

GPU0

CPU

GPU1
Tasks execution on a heterogeneous node

- CPU
- GPU0
- GPU1

Handles dependencies
Handles scheduling: depends on user chosen scheduler
Handles data consistency

But what is a task precisely?

GEMM
SYRK
TRSM
POTRF

Exloiting multi-level parallelism on Intel KNL
Tasks execution on a heterogeneous node
Tasks execution on a heterogeneous node

Handles dependencies
Handles scheduling: depends on user chosen scheduler
Handles data consistency

But what is a task precisely?

GEMM
SYRK
TRSM
POTRF

Exloiting multi-level parallelism on Intel KNL
Tasks execution on a heterogeneous node

CPU

GPU0

CPU

GPU1

Handles dependencies
Handles scheduling: depends on user chosen scheduler
Handles data consistency

But what is a task precisely?

GEMM
SYRK
TRSM
POTRF

Exloiting multi-level parallelism on Intel KNL
Tasks execution on a heterogeneous node

Handles dependencies
Handles scheduling: depends on user chosen scheduler
Handles data consistency

But what is a task precisely?

GEMM
SYRK
TRSM
POTRF

Exloiting multi-level parallelism on Intel KNL
Tasks execution on a heterogeneous node

- Handles dependencies
Tasks execution on a heterogeneous node

- Handles dependencies
Tasks execution on a heterogeneous node

- Handles dependencies

GEMM
SYRK
TRSM
POTRF

Exloiting multi-level parallelism on Intel KNL
Tasks execution on a heterogeneous node

- Handles dependencies
- Handles scheduling: depends on user chosen scheduler
Tasks execution on a heterogeneous node

Handles dependencies
Handles scheduling: depends on user chosen scheduler
Tasks execution on a heterogeneous node

- Handles dependencies
- Handles scheduling: depends on user chosen scheduler
- Handles data consistency

But what is a task precisely?

GEMM
SYRK
TRSM
POTRF

< Terry.Cojean@inria.fr >

Exloiting multi-level parallelism on Intel KNL
Tasks execution on a heterogeneous node

- Handles dependencies
- Handles scheduling: depends on user chosen scheduler
- Handles data consistency
- But what is a task precisely?
Outline

1. The StarPU runtime system
   - Aim of runtime systems
   - The Sequential Task Flow (STF) model

2. Paradigm evolution: Parallel tasks in StarPU

3. Resource aggregation experiments on the Intel KNL
   - Machine and kernels overview
   - Parallel tasks performance
Overly simplistic global view of the StarPU Model

Tasks are pushed to the runtime
 Tasks are spread on resources according to the scheduler (e.g. HEFT, WS policies)
 Workers (CPU cores/GPU/...) execute their tasks
Slight changes to the model for efficient KNL usage

- Huge amount of cores
- Is scheduling independently on 64 cores pertinent?
- We might be fine now but what about a KNL like machine with even more cores?

- We could increase machine efficiency
- Shared L2 cache means a need for shared work
- Independent tasks such as StarPU’s are bad for work sharing
Solution: adapting the machine to the problem

- Same tasks pushed to StarPU
- Like before, schedulers spread tasks to resources
- The only differences are
  - Somehow we view a group of cores together
  - Tasks are parallel: we can execute them on multiple resources
Solution: adapting the machine to the problem

Same tasks pushed to StarPU
Like before, schedulers spread tasks to resources
The only differences are
- Somehow we view a group of cores together
- Tasks are parallel: we can execute them on multiple resources

These parallel tasks are executed on top of the resources by another runtime (e.g., OpenMP)
The way it works internally is a black-box to StarPU
StarPU only ensures that the internal runtime uses the resources allocated to it.
Parallel tasks model

**Idea 1:** adapting the machine to the problem.

**Idea 2:** delegate some of the work to another runtime.

Parallel tasks model: precisely what we need

- Theoretical model proposed in the 90s
- Scarcely used in practice

A task has a new parameter: the amount of threads/resources allocated to it.

Here:

- Task parallelism: StarPU
- Internal parallelism: can be anything.
  Good candidate: OpenMP (Parallel BLAS, efficient cache reuse, ...)

\(<\text{Terry.Cojean@inria.fr}>\)
Outline

1. The StarPU runtime system
   - Aim of runtime systems
   - The Sequential Task Flow (STF) model

2. Paradigm evolution: Parallel tasks in StarPU

3. Resource aggregation experiments on the Intel KNL
   - Machine and kernels overview
   - Parallel tasks performance
Experimental setup

Machine
- All experiments were run on KNL 7210
- Configuration was SNC-4 cached
- Preliminary results show the same pattern on KNL 7230

Software
- We use Intel compiler and MKL 17
- Chameleon: a Dense Linear Algebra software on top of multiple runtime systems developed by HiePACS team.
- Here we use Chameleon on top of the StarPU runtime system.

Methodology
- First, study and understand the kernel performances
- Secondly, benchmark Chameleon’s Cholesky factorization with StarPU using different core group size
  - Create bigger resources
  - Run the whole execution on these big resources
Here are the problems we will look at

1. What is the best tile size performance wise?
2. What is the impact of parallelism loss (bigger tile size)?
Cholesky kernels performance with one core on KNL 7210

Exloiting multi-level parallelism on Intel KNL
Cholesky kernels parallel efficiency on KNL 7210

- POTRF
- TRSM
- SYRK
- GEMM

Parallel Efficiency vs. Number of Threads

- Size: 480, 960, 1920, 2880

Exloiting multi-level parallelism on Intel KNL
Comparison with MKL and Plasma

Exloiting multi-level parallelism on Intel KNL
Understanding results: actual DGEMM kernels performance

Matrix Size 34K, performance of DGEMM kernels of Cholesky Factorization

![Graph showing performance comparison]

Configuration: Chameleon (Tile Size: 480)

Configuration: pt-Chameleon 4x16 (Tile Size: 2880)

Exloiting multi-level parallelism on Intel KNL
Conclusion

On the KNL
- Feed it lots of work
- Beware of high contention
- SNC-4 is fairly good performance wise
- Cache mode is fine for BLAS3

On the use of parallel tasks
- Less resources, work sharing between cores
- Allows to reduce various contentions, very effective on KNL
- Allows to increase task granularity for better performance
- We can beat MKL, by using MKL...

Ongoing and Future Work
- Adapt group size at different phases of the execution
- Profit from group size heterogeneity similarly to CPU/GPU
- Integrate this model into OpenMP?