Getting started with StarPU

Cédric Augonnet
Nathalie Furmento
Samuel Thibault
Raymond Namyst

INRIA Bordeaux, LaBRI, Université de Bordeaux

The RUNTIME Team

Doing Parallelism for centuries!
The RUNTIME Team

Research directions

• High Performance Runtime Systems for Parallel Architectures
  • “Runtime Systems perform dynamically what cannot be not statically”

• Main research directions
  • Exploiting shared memory machines
    – Thread scheduling over hierarchical multicore architectures
    – Task scheduling over accelerator-based machines
  
  • Communication over high speed networks
    – Multicore-aware communication engines
    – Multithreaded MPI implementations
  
  • Integration of multithreading and communication
    – Runtime support for hybrid programming
  
• See http://runtime.bordeaux.inria.fr/ for more information
The StarPU runtime system
Motivation
Accelerator-based architectures

• Main Challenges
  • Dynamically schedule tasks on all processing units
    – See a pool of heterogeneous cores
    – Scheduling ≠ offloading
  • Avoid unnecessary data transfers between accelerators
    – Need to keep track of data copies
Motivation
The need for runtime systems

• “do dynamically what can’t be done statically”

• Typical duties
  • Task scheduling
  • Memory management

• Compilers and libraries generate (graphs of) parallel tasks
  • Additional information is welcome!
The StarPU runtime system
Memory Management

• StarPU provides a **Virtual Shared Memory** subsystem
  • Weak consistency
  • Replication
  • Single writer
  • High level API
    – Partitioning filters

• Input & output of tasks = reference to VSM data

HPC Applications

Parallel Compilers

Parallel Libraries

StarPU

Drivers (CUDA, OpenCL)

CPU  GPU  ...
The StarPU runtime system

Task scheduling

• **Tasks =**
  - Data input & output
    – Reference to VSM data
  - Multiple implementations
    – E.g. CUDA + CPU implementation
  - Dependencies with other tasks
  - Scheduling hints

• StarPU provides an Open Scheduling platform
  - Scheduling algorithm = plug-ins
The StarPU runtime system
Development context

• History
  • Started about 2 years ago
  • StarPU main core ~ 20k lines of code
  • Written in C
  • 3 core developers
    – Cédric Augonnet, Samuel Thibault, Nathalie Furmento

• Open Source
  • Released under LGPL
  • Sources freely available
    – svn repository and nightly tarballs
    – See http://runtime.bordeaux.inria.fr/StarPU/
  • Open to external contributors
The StarPU runtime system

Supported platforms

• Supported architectures
  • Multicore CPUs (x86, PPC, ...)
  • NVIDIA GPUs
  • OpenCL devices (eg. AMD cards)
  • Cell processors (experimental)

• Supported Operating Systems
  • Linux
  • Mac OS
  • Windows
Test case
Mixing PLASMA and MAGMA
Mixing PLASMA and MAGMA with StarPU

• PLASMA BLAS
  • Rely on vendors' BLAS

• MAGMA BLAS
  • Autotuned kernels
  • Rely on CUBLAS
  • Provides new kernels

• PLASMA
  • Tile algorithms
  • Dynamically scheduled tasks
    – Alternative to PLASMA's scheduler (QUARK)
    – Extend it for GPUs
Mixing PLASMA and MAGMA with StarPU

• Methodology (~a week of work)
  • Describe data layout
    – Register all tiles to StarPU

• Create tasks
  • Codelets: Multi-versionned functions
    – PLASMA kernels on CPU
    – MAGMA kernels on GPU
    – Access registered tiles

• Dynamically submit a DAG of tasks
  • Automatic dependencies
  • No mapping decision
Mixing PLASMA and MAGMA with StarPU

- Cholesky decomposition
  - 5 CPUs (Nehalem) + 3 GPUs (FX5800)
  - Efficiency > 100%

![Graphs showing performance and speedup with different combinations of CPUs and GPUs.](Image)
Mixing PLASMA and MAGMA with StarPU

- Cholesky decomposition
  - 5 CPUs (Nehalem) + 3 GPUs (FX5800)
  - Efficiency > 100%
Mixing PLASMA and MAGMA with StarPU

- Memory transfers during Cholesky decomposition

![Graph showing memory transfers during Cholesky decomposition](image)

The graph illustrates the total amount of data transfers for different matrix sizes, comparing scenarios with and without support for locality. The graph shows a significant reduction in transfers, approximately a 2.5x decrease.
Installing StarPU
Downloading StarPU

Getting sources

• Access SVN sources (via svn or https)
  • `svn checkout svn://scm.gforge.inria.fr/svn/starpu/trunk`
  • `svn checkout --username anonsvn https://scm.gforge.inria.fr/svn/starpu/trunk`
  • Password : anonsvn
  • Requires : Autoconf (>= 2.60) & Automake
  • Run `./autogen.sh` to generate a `./configure` script

• Nightly Tarball
  • `http://starpu.gforge.inria.fr/testing/starpu-nightly-latest.tar.gz`
  • Useful when autotools are not available (or recent enough)
Optional dependency with hwloc

Topology discovery external library

• hwloc
  • Topology discovery library
  • Portable!
  • Initially developed in the RUNTIME team

• StarPU & hwloc
  • Not mandatory but strongly recommended
    – Increase performance
    – Topology aware scheduling

• Getting hwloc
  • Available in major distributions and for most OS
  • Download from http://www.open-mpi.org/software/hwloc
Configuring and Compiling StarPU

• Standard installation procedure
  • ./configure
  • or ./configure --prefix=$HOME/StarPU/
  • --enable-verbose to get some debug messages
  • --help to get a summary of options

• Compiling
  • make -j
  • make install

• Sanity checks
  • make check
Installing StarPU

- make install

- Environment variables (typically added into ~/.bashrc)
  - export STARPUDIR=<StarPU's installation directory>
    export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${STARPUDIR}/lib/
    export PATH=${PATH}:${STARPUDIR}/bin/
  - export PKG_CONFIG_PATH=${PKG_CONFIG_PATH}:${STARPUDIR}/lib/pkgconfig/

- Using pkg-config
  - pkg-config --cflags libstarpu : compiler flags
  - pkg-config --libs libstarpu : linker flags
  - Example for libs
    - -L/home/gonnet/StarPU/trunk/target/lib -L/usr/local/cuda/lib64/ -lstarpu -lcuda
Installing StarPU (2)

• When StarPU is used for the first time
  • $HOME/.StarPU/ is created
    – Contains performance models
  • Buses are benchmarked when StarPU is launched for the time
    – May take a few minutes!
      – Faster if hwloc is installed
    – Only once per user and per machine
A trivial example: scaling a vector
Scaling a vector
Launching StarPU

• Makefile flags
  • CFLAGS += $$\$(\text{pkg-config} \text{ --cflags} \text{ libstarpu})$
  • LDFLAGS+=$$\$(\text{pkg-config} \text{ --libs} \text{ libstarpu})$

• Headers
  • \#include <starpu.h>

• (De)Initialize StarPU
  • starpu_init(NULL);
  • starpu_shutdown();
Scaling a vector

Data registration

• Register a piece of data to StarPU
  • float array[NX];
    for (unsigned i = 0; i < NX; i++)
      array[i] = 1.0f;

  starpu_data_handle vector_handle;
  starpu_vector_data_data_register(&vector_handle, 0,
      array, NX, sizeof(vector[0]));

• Unregister data
  • starpu_data_unregister(vector_handle);
Scaling a vector
Defining a codelet

• CPU kernel

```c
void scal_cpu_func(void *buffers[], void *cl_arg)
{
    struct starpu_vector_interface_s *vector = buffers[0];

    unsigned n = vector->nx;
    float *val = (float *)vector->ptr;

    float *factor = cl_arg;
    for (int i = 0; i < n; i++)
        val[i] *= *factor;
}
```
Scaling a vector
Defining a codelet (2)

- CUDA kernel (compiled with nvcc, in a separate .cu file)

```c
__global__ void vector_mult_cuda(float *val, unsigned n, float factor)
{
    for(unsigned i = 0 ; i < n ; i++) val[i] *= factor;
}
```

```c
extern "C" void scal_cuda_func(void *buffers[], void *cl_arg)
{
    float *factor = (float *)cl_arg;
    struct starpu_vector_interface_s *vector = buffers[0];
    unsigned n = vector->nx;
    float *val = (float *)vector->ptr;

    vector_mult_cuda<<<1,1>>>(val, n, *factor);
    cudaThreadSynchronize();
}
```
Scaling a vector
Defining a codelet (3)

• Codelet = multi-versionned kernel
  • Function pointers to the different kernels
  • Number of data managed by StarPU

```c
starpu_codelet scal_cl = {
  .where = STARPU_CPU | STARPU_CUDA,
  .cpu_func = scal_cpu_func,
  .cuda_func = scal_cuda_func,
  .nbuffers = 1
};
```
Scaling a vector

Defining a task

- Define a task that scales the vector by a constant

```c
struct starpu_task *task = starpu_task_create();
task->cl = &scal_cl;

task->buffers[0].handle = vector_handle;
task->buffers[0].mode = STARPU_RW;

float factor = 3.14;
task->cl_arg = &factor;
task->cl_arg_size = sizeof(factor);

starpu_task_submit(task);
starpu_task_wait(task);
```
Data Management
StarPU data interfaces

StarPU data coherency protocol

• Memory nodes
  • Each worker is associated to a node
  • Multiple workers may share a node

• Data coherency
  • Keep track of replicates
  • Discard invalid replicates

• MSI coherency protocol
  • M : Modified
  • S : Shared
  • I : Invalid

\[ A = A + B \]
StarPU data interfaces
StarPU data coherency protocol

- Memory nodes
  - Each worker is associated to a node
  - Multiple workers may share a node

- Data coherency
  - Keep track of replicates
  - Discard invalid replicates

- MSI coherency protocol
  - M : Modified
  - S : Shared
  - I : Invalid

\[
A = A + B
\]
StarPU data interfaces

• Each piece of data is described by a structure
  • Example: vector interface
    ```c
    struct starpu_vector_interface_s {
      unsigned nx;
      unsigned elemsize;
      uintptr_t ptr;
    }
    ```
  • StarPU ensures that interfaces are coherent

• StarPU tasks are passed pointers to these interfaces
  • Coherency protocol is independent from the type of interface
StarPU data interfaces

• Various interfaces are available
  • Variable, Vector, Matrix, CSR ....

• Defining a new interface
  • C structure
  • struct starpu_data_interface_ops_t
    – Transfer between nodes
    – Allocate on node
    – Get data size …
StarPU data interfaces

• Registering a piece of data
  • Generic method
    starpu_data_register(starpu_data_handle *handleptr,
                         uint32_t home_node, void *interface,
                         struct starpu_data_interface_ops_t *ops);
  • Wrappers are available for existing interfaces
    starpu_variable_data_register(starpu_data_handle *handle,
                                   uint32_t home_node,
                                   uintptr_t ptr, size_t elemsize);
    starpu_vector_data_register(starpu_data_handle *handle,
                                 uint32_t home_node,
                                 uintptr_t ptr, uint32_t nx, size_t elemsize);
    starpu_csr_data_register(starpu_data_handle *handle, uint32_t home_node,
                              uint32_t nnz, uint32_t nrow, uintptr_t nzval,
                              uint32_t *colind,
                              uint32_t *rowptr, uint32_t firstentry, size_t elemsize);
StarPU data interfaces
Manipulating registered data

• Use handle to manipulate a registered piece of data
  • Task description
  • starpu_data_unregister(handle)

• Interactions between StarPU and the application are possible
  • int foo = 42;
    starpu_variable_register(&handle, 0, &foo, sizeof(foo));
    ...
    starpu_data_sync_with_mem(handle, STARPU_RW);
    foo = 12;
    starpu_data_release(handle);
    ...
  • See starpu_data_sync_with_mem_non_blocking for an asynchronous method
Task Management
Task management

Codelets

- struct starpu_codelet_t

- Describe multi-versionned kernels
  - Where can the kernels be executed?
    - eg. STARPU_CPU | STARPU_CUDA | STARPU_OPENCL
  - Per-architecture implementation
    - Function pointers
      - Running on the host!
        - eg. use CUDA runtime API
    - Common interface
      - void cpu_func(void *buffers[], void *cl_arg);
  - Specify the number of buffers accessed by the codelet
    - Only data managed by StarPU
    - Use cl_arg for constant arguments
  - Optional: Performance model
Task management

Performance models

- `struct starpu_perfmodel_t`

- Different types of performance models
  - `STARPU_COMMON`
    - a single model + relative speedups
  - `STARPU_PER_ARCH`
    - per processing-unit performance models
  - `STARPU_REGRESSION_BASED`
    - Specify the type of regression with the `starpu_regression_model_t` structure
      - Online Linear models (eg. a*size^3)
      - Offline Non-linear models (eg. a*size^3 + b)
  - `STARPU_HISTORY_BASED`
    - History based performance models
    - Updated every time a task is executed
    - For regular applications

- Automatically calibrated (`STARPU_CALIBRATE` env. variable)
Task management

Performance models

• Using performance models is almost transparent
  • Automatically calibrated by StarPU
  • Associate each codelet with a unique identifier (symbol)

```c
struct starpu_permmodel_t sgemm_model = {
    .type = STARPU_HISTORY_BASED,
    .symbol = "sgemm"
};

starpu_codelet sgemm_cl = {
    .where = STARPU_CPU|STARPU_CUDA,
    .cpu_func = cpu_sgemm,
    .cuda_func = cuda_sgemm,
    .nbuffers = 3,
    .model = &sgemm_model
};
```
Task management

The task structure

• struct starpu_task

• Task description
  • struct starpu_codelet_t *cl
    – void *cl_arg : constant argument passed to the codelet
  • Buffers array (accessed data + access mode)
    task->buffers[0]->handle = vector_handle;
    task->buffers[0]->mode = STARPU_RW;
  • void (*callback_func)(void *);
    – void *callback_arg;
    – Should not be a blocking call!
  • Extra hints for the scheduler
    – eg. priority level
Task management

Task API

- Create tasks
  - Dynamically allocated by starpu_task_create
  - Otherwise, initialized by starpu_task_init

- Submit a task
  - starpu_task_submit(task)
    - blocking if task->synchronous = 1

- Wait for task termination
  - starpu_task_wait(task);
  - starpu_task_wait_for_all();

- Destroy tasks
  - starpu_task_destroy(task);
    - automatically called if task->destroy = 1
  - starpu_task_deinit(task);
Task management

Explicit task dependencies

- Submit tasks within task callbacks

- Task dependencies
  - starpu_task *deps[2] = {taskA, taskB}
    void starpu_task_declare_deps_array(taskC, 2, &deps);
  - TaskC depends on taskA and taskB
  - Must be declared prior to the submission of taskC!

- Tag dependencies
  - task->tag_id logically identifies a task if task->use_tag is set
    - taskA->tag_id = 0x2000;
    - taskB->tag_id = 0x42;
    - starpu_tag_declare_deps(0x42, 0x2000)
      - TaskB depends on taskA
    - starpu_tag_wait(0x42)
      - Wait for taskB (taskB must have been submitted)
Task management
Implicit task dependencies

- StarPU can discover data dependencies automatically
  - Sequential data consistency
    - Match the behaviour of sequential code
  - Example: f1(Ar); f2(Ar); g1(Arw); g2(Arw); h1(Ar); h2(Ar);
    - f1 and f2 can be done in parallel
    - g1 depends on \{f1,f2\}
    - g2 depends on g1
    - h1 and h2 can be done in parallel, but depends on g2

- Enabled by default for all data handles
  - \texttt{void starpu_data_set_default_sequential_consistency_flag(unsigned flag);} 

- Per-handle parameter
  - eg. RW access on accumulator should not imply a dependency
  - \texttt{void starpu_data_set_sequential_consistency_flag(starpu_data_handle handle, unsigned flag);}
Task management
Implicit task dependencies (2)

• Right-Looking Cholesky decomposition (from PLASMA)
  • For \( k = 0 \ldots \text{tiles} - 1 \)
    
    \[
    \begin{align*}
    &\{ \\
    &\quad \text{POTRF}(A[k,k]) \\
    &\quad \text{for } (m = k+1 \ldots \text{tiles} - 1) \\
    &\quad \text{TRSM}(A[k,k], A[m,k]) \\
    &\quad \text{for } (n = k+1 \ldots \text{tiles} - 1) \\
    &\quad \{ \\
    &\quad \text{SYRK}(A[n,k], A[n,n]) \\
    &\quad \text{for } (m = k+1 \ldots \text{tiles} - 1) \\
    &\quad \text{GEMM}(A[m,k], A[n,k]) \\
    &\quad \} \\
    &\} \\
    \end{align*}
    \]
Running the application
Running the application

Environment variables

- Select the number of processing units
  - STARPU_NCPUS
  - STARPU_NCUDA
  - STARPU_NOPENCL

- Select the scheduling policy
  - Run with STARPU_SCHED=help to get the different options
  - STARPU_SCHED=greedy (default)
  - STARPU_SCHED=ws
  - STARPU_SCHED=dm (use task performance models)
  - STARPU_SCHED=dmda (task + data transfer models)
    _ STARPU_PREFETCH=1 is also recommended

- Calibrate performance models
  - STARPU_CALIBRATE=1
  - STARPU_CALIBRATE=2 (also erase existing models)
  - May takes a few runs to be fully calibrated
    _ Possibly calibrate models on small problems!
Performance analysis
Offline performance analysis
Generate execution traces

• The FxT library
  • Low overhead tracing facility
  • Get FxT sources
    – `cvs -d :pserver:anonymous@cvs.sv.gnu.org:/sources/fkt co FxT`
  • Install FxT
    – `./bootstrap`
    `./configure --prefix=$FXT_INSTALL_DIR`
    `make; make install`

• Configure StarPU to generate traces
  • `./configure --with-fxt=$FXT_INSTALL_DIR`

• Run the application
Offline performance analysis

Visualize execution traces

- Generate a Pajé trace
  - A file of the form /tmp/prof_file_user_<your login> should have been created
  - Call fxt_tool -i /tmp/prof_file_user_yourlogin
    - A paje.trace file should be generated in current directory

- Vite trace visualization tool
  - Freely available from http://vite.gforge.inria.fr/ (open source !)
  - vite paje.trace

![Image of Vite tool]

2 Xeon cores
Quadro FX5800
Quadro FX4600