A performance-oriented comparative study of the Chapel high-productivity language to conventional programming environments

Guillaume Helbecque\(^1\) (speaker), Jan Gmys\(^1\), Tiago Carneiro\(^2\), Nouredine Melab\(^1\), Pascal Bouvry\(^3\)

\(^1\)Université de Lille  
CNRS/CRIStAL  
Inria Lille Nord-Europe  
France

\(^2\)Université du Luxembourg  
FSTM  
Luxembourg

\(^3\)Université du Luxembourg  
DCS-FSTM/SnT  
Luxembourg
Background

- Heterogeneity (CPU-GPU) and size of modern supercomputers (millions of cores) [1]:

<table>
<thead>
<tr>
<th>Rank</th>
<th>System</th>
<th>Cores</th>
<th>Rmax (TFlop/s)</th>
<th>Rpeak (TFlop/s)</th>
<th>Power (kW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Supercomputer Fugaku - Supercomputer Fugaku, A60X 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science, Japan</td>
<td>7,630,848</td>
<td>442,010.0</td>
<td>537,212.0</td>
<td>29.699</td>
</tr>
<tr>
<td>2</td>
<td>Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SG/Oak Ridge National Laboratory, United States</td>
<td>2,414,592</td>
<td>148,600.0</td>
<td>200,794.9</td>
<td>10.096</td>
</tr>
<tr>
<td>3</td>
<td>Sierra - IBM Power System AC922, IBM POWER9 22C 3.10GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NSA/LNL, United States</td>
<td>1,572,480</td>
<td>94,660.0</td>
<td>125,712.0</td>
<td>7.438</td>
</tr>
</tbody>
</table>

Top 3 of the Top500 bi-annual ranking (Nov. 2021)

- Emergence of high-productivity programming languages [2]:
  - Chapel, X10, Fortress, etc.
Motivations and objectives

- Compare Chapel to conventional parallel programming libraries, in terms of performance...
  - on both shared- and distributed-memory systems;

- Illustrate the programming effort in each parallel environment... and provide a sense of "productivity";

- Provide a useful data point using shared- and distributed-memory multi-core systems for supercomputer programmers...
  - through a well-known and complete parallel application.
Outline

- Formulation of the test-case
- Parallel programming environments and implementations
  - OpenMP
  - Chapel
  - MPI
  - Hybrid MPI+OpenMP
- Experimental evaluation
  - Testbed
  - Shared-memory experiments
  - Distributed-memory experiments
  - Parallel overheads
- Conclusions and future works
Test-case: the Mandelbrot set

It is defined as the set of complex numbers \( c = a + ib \in \mathbb{C} \) such that the sequence \( (z_n)_{n \in \mathbb{N}} \subset \mathbb{C} \) defined by

\[
z_0 = 0, \quad z_{n+1} = z_n^2 + c,
\]
remains bounded in \( \mathbb{C} \).

Pseudo implementation of the Mandelbrot set computation

```python
function Compute_pixel(a, b):
    x = y = 0;
    n = 0;
    while \( x^2 + y^2 < 4 \) and \( n < N \) do
        t = x;
        x = \( x^2 - y^2 + a \);
        y = 2tx + b;
        n = n + 1;
    end
    I(a, b) = n/N;
end
```

- Embarrassingly parallel application, due to the **independency between pixels**;
- **Domain decomposition** method, **along the lines**;
- **Static decomposition**;
- Lines are mapped in **round-robin fashion**.

Monochrome Mandelbrot set
Open Multi-Processing (OpenMP)

• OpenMP is an application programming interface;

• It is designed for:
  ➢  **Ease of programming**;
  ➢  **High performance**, and;
  ➢  **Portability**;
  … only on shared-memory systems;

• OpenMP supports a **multithreaded execution**, through a fork-join model;

• It provides **simple high-level constructs**, for work-sharing among threads for example;

• For more details [3]: https://www.openmp.org/.

---

Pseudo OpenMP implementation of the Mandelbrot set computation

```plaintext
function Compute_image_omp() :
pragma omp parallel for schedule(static, 1):
for a = 0 to nb_lines do
  for b = 0 to nb_columns do
    Compute_pixel(a, b);
end
end
```

PMAM'22, April 2–6, 2022
Cascade High Productivity Language (Chapel)

- Chapel is a **PGAS-based language**;

- It is designed for:
  - **Ease of programming**;
  - **High performance**, and;
  - **Portability**;

- Chapel supports a **multithreaded execution model** and allows:
  - **Data/task parallelism**;
  - **Locality control**, etc;

- Here Chapel follows the **SPMD execution model**;

- For more details [4]: https://chapel-lang.org/.

Pseudo Chapel implementation of the Mandelbrot set computation
**Message Passing Interface (MPI)**

- Message-passing **application programming interface**;

- MPI is widely used in the academic and industrial areas:
  - Portability;
  - High performance, and;
  - Standardization;

- MPI defines a **communication protocol among processes** running on distributed-systems:
  - Point-to-point or **two-sided**;
  - One-sided;

- Here MPI follows the **SPMD execution model**;

- For more details [5]: [https://www.open-mpi.org/](https://www.open-mpi.org/).

---

```c
1 for l = rank to nb_lines by commsize do
2     Compute_line(l);
3     if rank ≠ 0 then
4         MPI_Send(*args*);
5     else
6         MPI_Recv(*args*);
7     end
8 end
```

Pseudo MPI two-sided implementation of the Mandelbrot set computation

```c
1 MPI_Win_create(win);
2 MPI_Fence();
3 for l = rank to nb_lines by commsize do
4     Compute_line(l);
5     if rank ≠ 0 then
6         MPI_PUT(*args*);
7     end
8 MPI_Fence();
9 end
```

Pseudo MPI one-sided implementation of the Mandelbrot set computation
Hybrid MPI+OpenMP

- OpenMP and MPI are complementary...
  - OpenMP is used for intranode parallelism;
  - MPI is used for distribution across nodes (1 MPI process/node).
Experimental environment

- Grid'5000 French national testbed [6]:
  - 6 computer nodes allocated;
  - 32 AMD EPYC 7301 CPUs @2.20GHz / nodes;
  - 25 Gbps Intel Ethernet Controller XXV710 network;

- gcc 10.2.1, Open MPI 4.1.0, OpenMP 4.5, Chapel 1.25.0;

- Compile options:
  - gcc –O2 optimization flag;
  - Chapel --fast optimization flag;

- Chapel multi-locale configuration:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHPL_RT_NUM_THREADS_PER_LOCALE</td>
<td>64</td>
</tr>
<tr>
<td>CHPL_TARGET_CPU</td>
<td>native</td>
</tr>
<tr>
<td>CHPL_HOST_PLATFORM</td>
<td>linux64</td>
</tr>
<tr>
<td>CHPL_LLVM</td>
<td>none</td>
</tr>
<tr>
<td>CHPL_COMM</td>
<td>gasnet</td>
</tr>
<tr>
<td>CHPL_COMM_SUBSTRATE</td>
<td>udp</td>
</tr>
<tr>
<td>GASNET_PSM_SPAWNER</td>
<td>ssh</td>
</tr>
</tbody>
</table>
Shared-memory experiments

- Fixed image size: 768x1024 pixels;
- **1 to 32 CPU cores (hyperthreading enabled, 2 threads/core);**
- N controls the granularity;
- 5 different implementations.

![Speed-up graphs](image)

Speed-up achieved by all five shared-memory implementations

PMAM'22, April 2–6, 2022
Distributed-memory experiments

- Fixed image size: 3840x5120 pixels;
- **1 to 192 CPU cores** (hyperthreading disabled);
- N controls the granularity;
- 5 different implementations.

![Graph showing speed-up for different N values](image)

Speed-up achieved by all five distributed-memory implementations
Parallel overheads

- **N=1** to highlight the communication overheads;
- Fixed image size: 768x1024 (shared) and 3840x5120 (distributed);
- Chapel relies by default on the *qthreads* tasking layer [7].

Computational overhead measured by all implementations in shared- (left) and distributed-memory (right) experiments

PMAM’22, April 2–6, 2022
Conclusions and future works

• Chapel...
  ✓ outperforms its counterparts in shared-memory, and;
  ✓ competes with hybrid MPI+OpenMP in distributed-memory.

• The *qthreads* default tasking layer of Chapel could explain these performances...
  ✓ although it seems to suffer from the lack of high-performance network between nodes.

• We plan to investigate...
  ✓ more complex benchmarks, involving message aggregation and data replication;
  ✓ other Chapel features, such as distributed iterators, specific data structures, etc.
Some references


Thank you for your attention!

Any question? Any remark?

guillaume.helbecque@univ-lille.fr