Best Practice Guide to Hybrid PaRSEC + OpenMP Programming

Version 1.0, 30th September 2018

Jakub Šístek, Reazul Hoque and George Bosilca

Table of Contents

1 Introduction

PaRSEC [1] is a generic framework for architecture-aware scheduling and management of micro-tasks on distributed many-core heterogeneous architectures. PaRSEC is the underlying runtime and dynamic scheduling engine of the DPLASMA numerical linear algebra library for dense matrices, among others. PaRSEC has been designed as a massively parallel distributed runtime system since its beginning around 2010.

A design assumption of PaRSEC was having a full control over the distributed memory resources, including memory, network, processor cores and accelerators. However, emerging hardware architectures as well as the need for reusing existing software require other modes of use of PaRSEC and call for smooth interoperability with other existing programming models.

OpenMP [2] is a clear candidate for investigating the interoperability with PaRSEC. It is widely used, supported by compilers and adopted by existing software.

Hybrid application programs using MPI + OpenMP are now common on large HPC systems. Details about the motivation of this combination can be found, e.g., in the related Best Practice Guide by INTERTWinE [3].

A hybrid programming model considered in this Best Practice Guide combines PaRSEC on the inter-node level with OpenMP used intra-node. In this approach, the tasks of PaRSEC are further parallelized by OpenMP. Such programming model is attractive especially for the possibility for reusing existing OpenMP software in an application based on distributed memory tasking. Data are interchanged on the inter-node level in an asynchronous way by the PaRSEC communication engine while being processed by OpenMP-parallel code.

Section 2 of this Best Practice Guide briefly describes PaRSEC and its interfaces. Section 3 covers binding of PaRSEC threads to hardware cores, and Section 4 focuses on the interplay of PaRSEC with OpenMP.

2 Interfaces of PaRSEC

Two main interfaces are currently available in the main PaRSEC distribution. The Parametrized Task Graph (PTG) and the Dynamic Task Discovery (DTD).

The PTG [4] interface expresses algorithms as parametrised graphs of tasks with dependencies. Unlike the standard (non-parametrised) Directed Acyclic Graph (DAG), PTG does not need access to the global DAG representation, and it is able to unfold the dependencies from any point of the graph. This feature is extremely important for maintaining scalability for very large numbers of nodes.

PTG is quite straightforward if the structure of the DAG is known at compile time. However, expressing dynamic algorithms, where the DAG is data dependent and cannot be expressed a-priori, may become challenging.

This limitation has been the main motivation for the recently developed DTD interface of PaRSEC [5]. This approach allows discovering and inserting tasks into the DAG at runtime, enhancing programmability for problems where the DAG is not available prior the computation. The resulting programming model is similar to the Sequential Task Flow (STF) of the StarPU runtime [6].

3 Binding of PaRSEC threads

PaRSEC has computation and communication threads. A specific number of computational threads is defined during the initialization of PaRSEC, by calling

```
context = parsec_init(cores, &argc, &argv);
```

If cores = -1, PaRSEC decides on the correct number of threads itself or takes the number from PaRSEC configuration file. Otherwise, users define the correct number at this call. If cores = 1, PaRSEC uses one computational thread per MPI process. If cores = 2, PaRSEC uses 2 threads per MPI process etc.

By default, PaRSEC binds its threads to the first core available on a node. This means that if cores = 1, PaRSEC will use the first core. To bind a thread of PaRSEC to a specific core, users need to use the parsec_bind option. The easiest way to express binding is through a file. Before diving into the interaction of PaRSEC with OpenMP, we will discuss the pure PaRSEC case and binding of PaRSEC threads to hardware cores. We follow the Wiki of PaRSEC [7].

We will explain the binding through examples. The tests were performed on one node of a cluster with two sockets, each equipped with 12-core processors. Core IDs 0-11 belong to the first socket while the IDs 12-21 to the second one. The correctness of the binding is checked using the *htop* utility.

We use an example of the block Cholesky factorization implemented using PaRSEC DTD interface as part of the INTERTWinE project. The individual kernels call parallel routines of the PLASMA library [8], which are parallelized using OpenMP tasks. The source code of the example is available at [9], in the "hybrid" branch.

Intel software stack (MPI, OpenMP, C compiler, etc.) 2018 was used in these runs. The latest version of PaRSEC and PLASMA libraries (September 2018) were used.

3.1 Binding 2 cores on one socket

If we want to bind two computational threads on the same socket we have to provide 2 IDs from the range 0-11. For example, let us choose the first two cores of the first socket, then the IDs are 0,1. We will create a binding file, say *binding.parsec*, with the content:

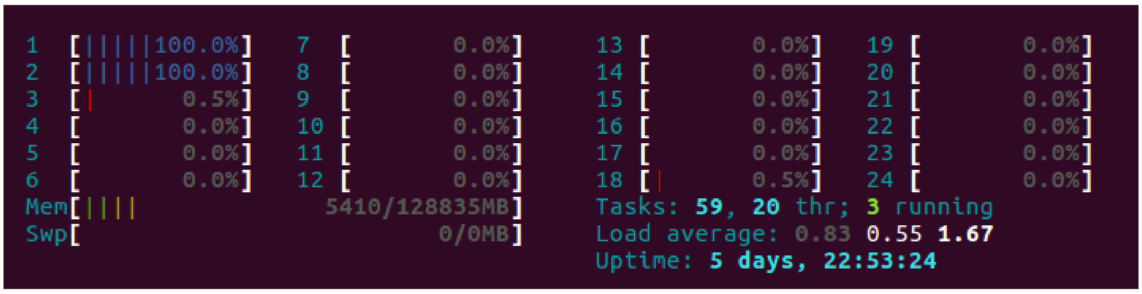
```
0,1
```

Here, the line number represents the rank we are trying to provide the binding information for. In this example, we are running shared-memory execution so the rank is 0 and the information needs to reside on the first line. Putting the information to any other line in the file will not work.

We run:

```
./test/test    dpotrf    --dim=20000    --nb=2000    --ib=250    --test=n    --parsec_bind
file:binding.parsec
```

The *htop* output shows:



```
1  [|||||100.0%]   7  [        0.0%]   13 [        0.0%]   19 [        0.0%]
2  [|||||100.0%]   8  [        0.0%]   14 [        0.0%]   20 [        0.0%]
3  [|        0.5%]   9  [        0.0%]   15 [        0.0%]   21 [        0.0%]
4  [         0.0%]   10 [        0.0%]   16 [        0.0%]   22 [        0.0%]
5  [         0.0%]   11 [        0.0%]   17 [        0.0%]   23 [        0.0%]
6  [         0.0%]   12 [        0.0%]   18 [|       0.5%]   24 [        0.0%]
Mem[||||              5410/128835MB]   Tasks: 59, 20 thr; 3 running
Swp[                          0/0MB]   Load average: 0.83 0.55 1.67
                                       Uptime: 5 days, 22:53:24
```

3.2 Binding 2 cores to different sockets

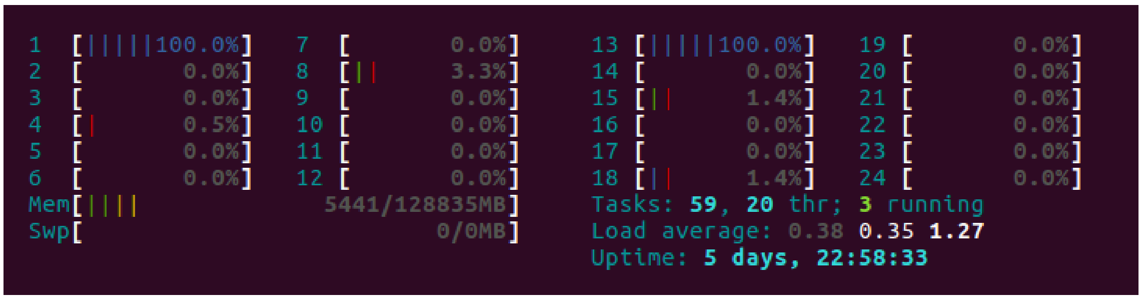
Let us now bind each of the PaRSEC threads to a different socket. In this case, the content of the *binding.parsec* file will change to:

```
0,12
```

The example is run as:

```
./test/test dpotrf --dim=20000 --nb=2000 --ib=250 --test=n --parsec_bind file:binding.parsec
```

Which results in the correct binding:



```
1  [|||||100.0%]   7  [        0.0%]    13 [|||||100.0%]   19 [        0.0%]
2  [        0.0%]   8  [||      3.3%]    14 [        0.0%]   20 [        0.0%]
3  [        0.0%]   9  [        0.0%]    15 [||      1.4%]   21 [        0.0%]
4  [|       0.5%]   10 [        0.0%]    16 [        0.0%]   22 [        0.0%]
5  [        0.0%]   11 [        0.0%]    17 [        0.0%]   23 [        0.0%]
6  [        0.0%]   12 [        0.0%]    18 [||      1.4%]   24 [        0.0%]
Mem[||||                5441/128835MB]    Tasks: 59, 20 thr; 3 running
Swp[                            0/0MB]    Load average: 0.38 0.35 1.27
                                          Uptime: 5 days, 22:58:33
```

3.3 Spawning 2 MPI processes with 1 PaRSEC thread each on different sockets

We will now use Intel MPI to spawn MPI processes and we will use a hostfile to bind each MPI process.

The content of the *mpi_hostfile* is as follows:

```
node0:0
node0:12
```

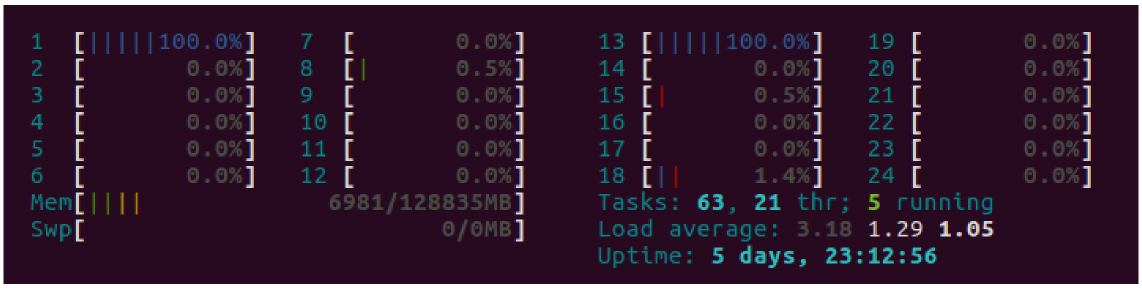
The *binding.parsec* file in this case is modified to:

```
0
12
```

Recall that each row corresponds to one rank. The example is run as:

```
mpirun -np 2 -hostfile mpi_hostfile ./test/test dpotrf --dim=20000 --nb=2000 --ib=250 --test=n --parsec_bind file:binding.parsec
```

and *htop* shows the correct binding:



```
1  [|||||100.0%]   7  [        0.0%]    13 [|||||100.0%]   19 [        0.0%]
2  [        0.0%]   8  [|       0.5%]    14 [        0.0%]   20 [        0.0%]
3  [        0.0%]   9  [        0.0%]    15 [|       0.5%]   21 [        0.0%]
4  [        0.0%]   10 [        0.0%]    16 [        0.0%]   22 [        0.0%]
5  [        0.0%]   11 [        0.0%]    17 [        0.0%]   23 [        0.0%]
6  [        0.0%]   12 [        0.0%]    18 [||      1.4%]   24 [        0.0%]
Mem[||||                6981/128835MB]    Tasks: 63, 21 thr; 5 running
Swp[                            0/0MB]    Load average: 3.18 1.29 1.05
                                          Uptime: 5 days, 23:12:56
```

We can see that computational cores of PaRSEC are correctly bound to cores 0 and 12 (1 and 13 in htop numbering).

Note that it is important to provide consistent binding in the *mpi_hostfile* and in the *binding.parsec* file. If the explicit binding was not provided for PaRSEC, both cores would bind to core 0.

3.4 Spawning 2 MPI processes with 1 PaRSEC thread each and explicit communication thread on different sockets

Apart of the computational threads, PaRSEC spawns a communication thread for each MPI process. In the previous example, the communication thread was not bound to any core and it was “floating”. One can get an explicit control over binding the communication thread of PaRSEC by adding another core ID at the end of the line in the *binding.parsec* file for each rank. In this case, we modify the *binding.parsec* of the previous example to:

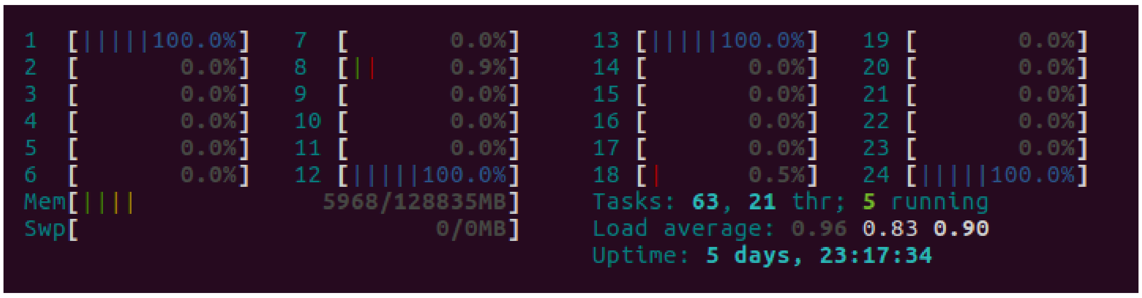
```
0,11
12,23
```

The last index on each row is used for binding the communication thread of PaRSEC spawned within the MPI process. Note that cores = 1 even in this case.

The program is executed by:

```
mpirun -np 2 -hostfile mpi_hostfile ./test/test dpotrf --dim=20000 --nb=2000 --ib=250 --test=n --parsec_bind file:binding.parsec
```

and *htop* shows



```
1   [|||||100.0%]   7   [         0.0%]     13 [|||||100.0%]   19 [         0.0%]
2   [         0.0%]   8   [||       0.9%]     14 [         0.0%]   20 [         0.0%]
3   [         0.0%]   9   [         0.0%]     15 [         0.0%]   21 [         0.0%]
4   [         0.0%]   10  [         0.0%]     16 [         0.0%]   22 [         0.0%]
5   [         0.0%]   11  [         0.0%]     17 [         0.0%]   23 [         0.0%]
6   [         0.0%]   12  [|||||100.0%]     18 [|        0.5%]   24 [|||||100.0%]
Mem[||||                   5968/128835MB]     Tasks: 63, 21 thr; 5 running
Swp[                             0/0MB]     Load average: 0.96 0.83 0.90
                                            Uptime: 5 days, 23:17:34
```

4 Combining PaRSEC threads with OpenMP threads

Once we understand how to bind PaRSEC threads within an MPI process, we can start with the different cases of the hybrid PaRSEC + OpenMP version.

4.1 Using 1 PaRSEC thread per node and all cores of the node for OpenMP threads

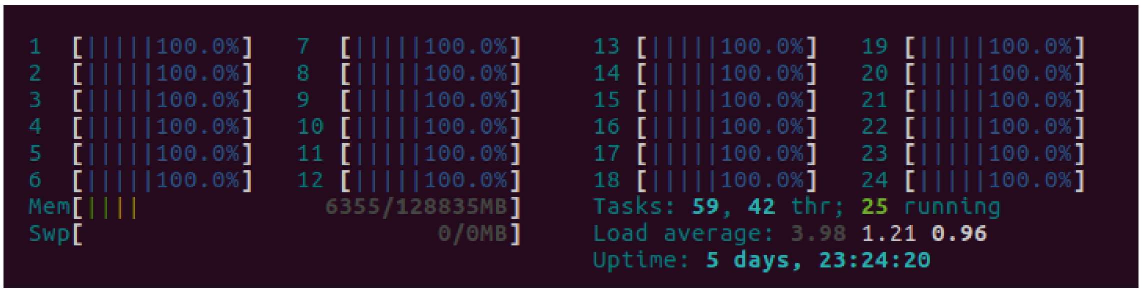
In this case, we aim to use one MPI process per node, one PaRSEC thread per MPI process and all the available cores for the OpenMP threads. We initialize PaRSEC with 1 core as the first argument of the call to the parsec_init function,

```
int cores = 1;
context = parsec_init(cores, &argc, &argv);
```

The code is executed as

```
OMP_NUM_THREADS=24 ./test/test dpotrf --dim=20000 --nb=2000 --ib=250 --test=n
```

Since we do not define the binding of the PaRSEC thread, it is bound, by default, to the first core of each node. The OMP_NUM_THREADS environment variable controls the number of OpenMP threads.



```
1  [|||||100.0%]   7  [|||||100.0%]    13 [|||||100.0%]   19 [|||||100.0%]
2  [|||||100.0%]   8  [|||||100.0%]    14 [|||||100.0%]   20 [|||||100.0%]
3  [|||||100.0%]   9  [|||||100.0%]    15 [|||||100.0%]   21 [|||||100.0%]
4  [|||||100.0%]   10 [|||||100.0%]    16 [|||||100.0%]   22 [|||||100.0%]
5  [|||||100.0%]   11 [|||||100.0%]    17 [|||||100.0%]   23 [|||||100.0%]
6  [|||||100.0%]   12 [|||||100.0%]    18 [|||||100.0%]   24 [|||||100.0%]
Mem[||||               6355/128835MB]    Tasks: 59, 42 thr; 25 running
Swp[                          0/0MB]    Load average: 3.98 1.21 0.96
                                         Uptime: 5 days, 23:24:20
```

4.2 Using 1 PaRSEC thread per socket and all cores of the socket for OpenMP threads

In this scenario, we aim to bind one MPI process per socket, run one PaRSEC thread in each of these processes and parallelize the PaRSEC tasks over all the cores of the socket by OpenMP. In this case, we will need to correctly combine the MPI process placement, the PaRSEC thread binding, and the OpenMP thread placement. The last one will be controlled over KMP_AFFINITY environment variable for the used Intel OpenMP runtime (It would be controlled by GOMP_CPU_AFFINITY variable for GNU OpenMP).

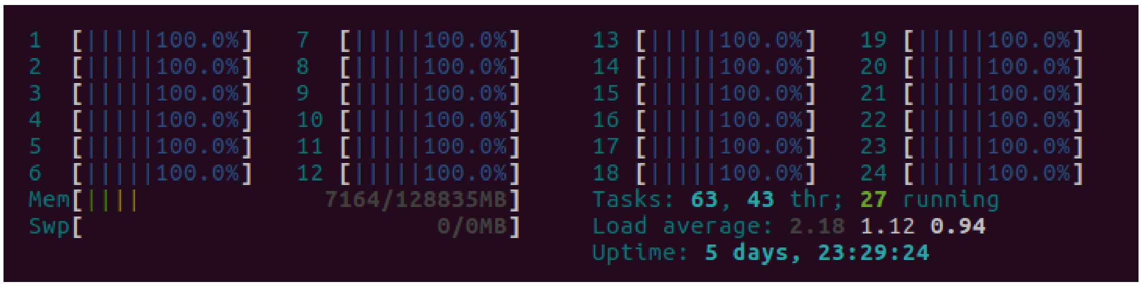
The *binding.parsec* file has two lines:

```
0
12
```

The *mpi_hostfile* is unchanged and the example is run as:

```
OMP_NUM_THREADS=12 KMP_AFFINITY=verbose,compact mpirun -np 2  -hostfile mpi_hostfile
./test/test   dpotrf   --dim=20000   --nb=2000   --ib=250   --test=n   --parsec_bind
file:binding.parsec
```

Now, *htop* shows:



```
  1  [|||||100.0%]   7  [|||||100.0%]    13 [|||||100.0%]   19 [|||||100.0%]
  2  [|||||100.0%]   8  [|||||100.0%]    14 [|||||100.0%]   20 [|||||100.0%]
  3  [|||||100.0%]   9  [|||||100.0%]    15 [|||||100.0%]   21 [|||||100.0%]
  4  [|||||100.0%]   10 [|||||100.0%]    16 [|||||100.0%]   22 [|||||100.0%]
  5  [|||||100.0%]   11 [|||||100.0%]    17 [|||||100.0%]   23 [|||||100.0%]
  6  [|||||100.0%]   12 [|||||100.0%]    18 [|||||100.0%]   24 [|||||100.0%]
  Mem[||||                7164/128835MB]    Tasks: 63, 43 thr; 27 running
  Swp[                            0/0MB]    Load average: 2.18 1.12 0.94
                                            Uptime: 5 days, 23:29:24
```

In this example, we rely on inheriting the OpenMP master thread from the MPI process.

4.3 Using 1 MPI process per socket and 2 PaRSEC computation threads in each of them

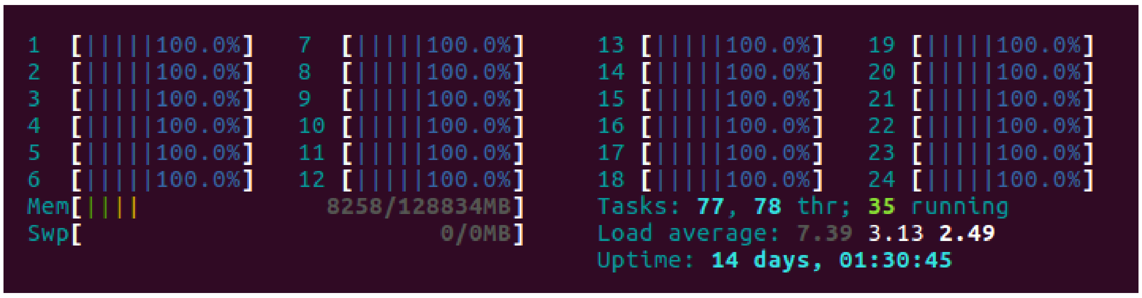
It is possible to combine MPI, PaRSEC and OpenMP in a more general way, with several PaRSEC computation threads within each MPI process. In this example, the number of PaRSEC threads is set to 2 (cores = 2). The *binding.parsec* file is changed to

```
0,6
12,18
```

and the code is executed as:

```
OMP_NUM_THREADS=6  KMP_AFFINITY=verbose,compact  mpirun  -np  2   -hostfile
mpi_hostfile  ./test/test  dpotrf  --dim=20000  --nb=2000  --ib=250  –test=n   --
parsec_bind file:binding.parsec
```

The outcome is checked in *htop*:



```
  1  [|||||100.0%]   7  [|||||100.0%]    13 [|||||100.0%]   19 [|||||100.0%]
  2  [|||||100.0%]   8  [|||||100.0%]    14 [|||||100.0%]   20 [|||||100.0%]
  3  [|||||100.0%]   9  [|||||100.0%]    15 [|||||100.0%]   21 [|||||100.0%]
  4  [|||||100.0%]   10 [|||||100.0%]    16 [|||||100.0%]   22 [|||||100.0%]
  5  [|||||100.0%]   11 [|||||100.0%]    17 [|||||100.0%]   23 [|||||100.0%]
  6  [|||||100.0%]   12 [|||||100.0%]    18 [|||||100.0%]   24 [|||||100.0%]
  Mem[||||                8258/128834MB]    Tasks: 77, 78 thr; 35 running
  Swp[                            0/0MB]    Load average: 7.39 3.13 2.49
                                            Uptime: 14 days, 01:30:45
```

4.4 Using 1 PaRSEC computation thread and 1 PaRSEC communication thread per socket and all the remaining cores for the OpenMP threads

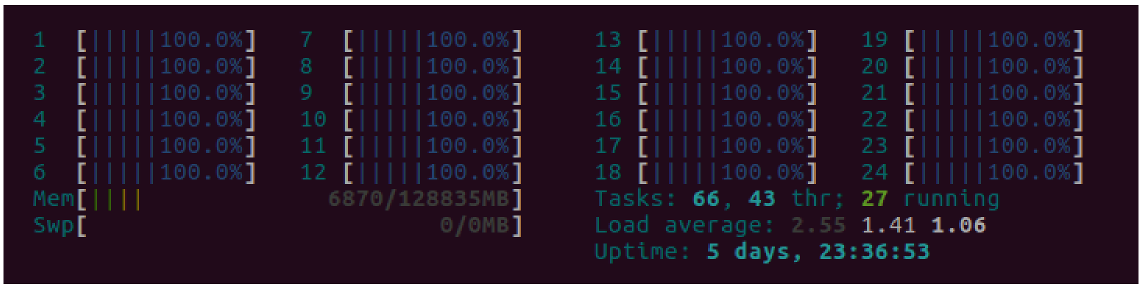
In the example above, the communication threads of PaRSEC were floating and assigned to any available core. In communication-intensive cases, it might be beneficial to dedicate one core to the communication thread of PaRSEC. This can be achieved by modifying the *binding.parsec* file to:

```
0,11
12,23
```

and executing the code with a reduced number of OpenMP threads:

```
OMP_NUM_THREADS=11 KMP_AFFINITY=verbose,compact mpirun -np 2 -hostfile mpi_hostfile
./test/test dpotrf --dim=20000 --nb=2000 --ib=250 --test=n --parsec_bind
file:binding.parsec
```

The *htop* outcome is similar as in the previous case, showing full utilization of the resources.



```
  1  [|||||100.0%]   7  [|||||100.0%]     13 [|||||100.0%]   19 [|||||100.0%]
  2  [|||||100.0%]   8  [|||||100.0%]     14 [|||||100.0%]   20 [|||||100.0%]
  3  [|||||100.0%]   9  [|||||100.0%]     15 [|||||100.0%]   21 [|||||100.0%]
  4  [|||||100.0%]   10 [|||||100.0%]     16 [|||||100.0%]   22 [|||||100.0%]
  5  [|||||100.0%]   11 [|||||100.0%]     17 [|||||100.0%]   23 [|||||100.0%]
  6  [|||||100.0%]   12 [|||||100.0%]     18 [|||||100.0%]   24 [|||||100.0%]
  Mem[||||               6870/128835MB]     Tasks: 66, 43 thr; 27 running
  Swp[                           0/0MB]     Load average: 2.55 1.41 1.06
                                            Uptime: 5 days, 23:36:53
```

5 References

[1] "PaRSEC webpage," [Online]. Available: http://icl.cs.utk.edu/parsec/. [Accessed September 2018].

[2] "OpenMP webpage," [Online]. Available: https://www.openmp.org/. [Accessed September 2018].

[3] INTERTWinE, "Best Practice Guide: MPI + OpenMP," [Online]. Available: https://www.intertwine-project.eu/sites/default/files/images/INTERTWinE_Best_Practice_Guide_MPI%2BOpenMP_1.2.pdf. [Accessed 22 Feb 2018].

[4] G. Bosilca, A. Bouteiller, A. Danalis, T. Herault, P. Lemariner and J. Dongarra, "DAGuE: A Generic Distributed DAG Engine for High Performance Computing," in *IEEE*, Anchorage, Alaska, USA, 2011.

[5] R. Hoque, T. Herault, G. Bosilca and J. Dongarra, "Dynamic Task Discovery in PaRSEC- A data-flow task-based Runtime," in *ScalA17: 8th Workshop on Latest Advances in Scalable Algorithms for LargeScale Systems*, Denver, CO, USA, 2017.

[6] E. Agullo, O. Aumage, M. Faverge, N. Furmento, F. Pruvost, M. Sergent and S. Thibault, "Achieving High Performance on Supercomputers with a Sequential Task-based Programming Model," *IEEE Transactions on Parallel and Distributed Systems (available online).*

[7] "PaRSEC Wiki," [Online]. Available: https://bitbucket.org/icldistcomp/parsec/wiki/Home. [Accessed September 2018].

[8] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, P. Wu, I. Yamazaki, A. YarKhan, M. Abalenkovs, N. Bagherpour, S. Hammarling, J. Šístek, D. Stevens, M. Zounon and S. D. Relton, "PLASMA: Parallel linear algebra software for multicore using OpenMP," *To appear in ACM Transactions on Mathematical Software,* 2018.

[9] "PLASMA+PaRSEC example," [Online]. Available: https://bitbucket.org/jakub_sistek/plasma_parsec. [Accessed September 2018].