Nekbone performance assessment report

Document Information

Reference Number	POP_AR_112 (Nekbone)
Author	Brian Wylie (JSC)
Contributor(s)	Ilya Zhukov (JSC)
Date	April 9, 2018

Notices: *The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 676553.*

Contents

1 Background

Applicants Name: Jing Gong
Institution: PDC, KTH, Stockholm, Sweden
Application Name: Nek5000/Nekbone
Programming Language: Fortran & C
Programming Model: MPI+OpenMP (MPI_THREAD_FUNNELED)
Source Code Available: yes
Performance study: check (audit)
User description: investigation of scalability
Application Description: Nek5000 (http://nek5000.mcs.anl.gov) is an open-source code for the simulation of incompressible flows. It is widely used in a broad range of applications, including the study of thermal hydraulics in nuclear reactor cores, the modeling of ocean currents and the simulation of combustion in mechanical engines. Nekbone is a proxy mini-app which captures the basic structure of the extensive Nek5000 software, exposing the principal computational kernel which solves a standard Poisson equation using the spectral element method with an iterative conjugate gradient (CG) solver with a simple preconditioner. The CORAL version of Nekbone 2.3.4.1 is implemented using MPI and OpenMP, with 8-byte floating-point values.
Input data: CORAL `data.rea`:

```
.true. = ifbrick                ! brick or linear geometry
512 512 1  = iel0,ielN,ielD (per processor)  ! range of number of elements per proc.
 9  12 3 = nx0,nxN,nxD           ! poly. order range for nx1
 1  1  1 = npx, npy, npz        ! processor distribution in x,y,z
 1  1  1 = mx, my, mz           ! local element distribution in x,y,z
```

(Actual processor and local element distributions determined during execution.)
Testcase Description: 1 MPI process with 64 OpenMP threads per compute node.
Machine Description: JUQUEEN IBM Blue Gene/Q with 28,672 compute nodes connected by proprietary 5D torus interconnect, and GPFS parallel filesystem. Each compute node has a single IBM PowerPC A2 16-core 1.6 GHz processor, with each core having four hardware threads, and runs a Linux-based microkernel OS. IBM BG compilers and MPI are provided.
Analysis tools: Scalasca/2.3.1 using Score-P/3.1 with hardware counters PAPI_TOT_INS and PAPI_TOT_CYC.

2 Setup

As described in the CORAL benchmark specification [https://asc.llnl.gov/CORAL-benchmarks/], a weak-scaling execution configuration is used, where the same amount of spectral elements (262,144) is given to each MPI process (and partitioned by 64 OpenMP threads). Executions were done on JUQUEEN with varying numbers of compute nodes. Around 390 MB of memory is required by each process.

An instrumented version of Nekbone for measurement was built using Score-P with IBM compiler instrumentation of source routines combined with OpenMP source construct preprocessing and MPI library interposition. Reported measurements were done employing a measurement filter consisting of twelve frequently executed short routines, identified by scoring a preliminary profile analysis report. (Measurement dilation with filtering still roughly 50%.)

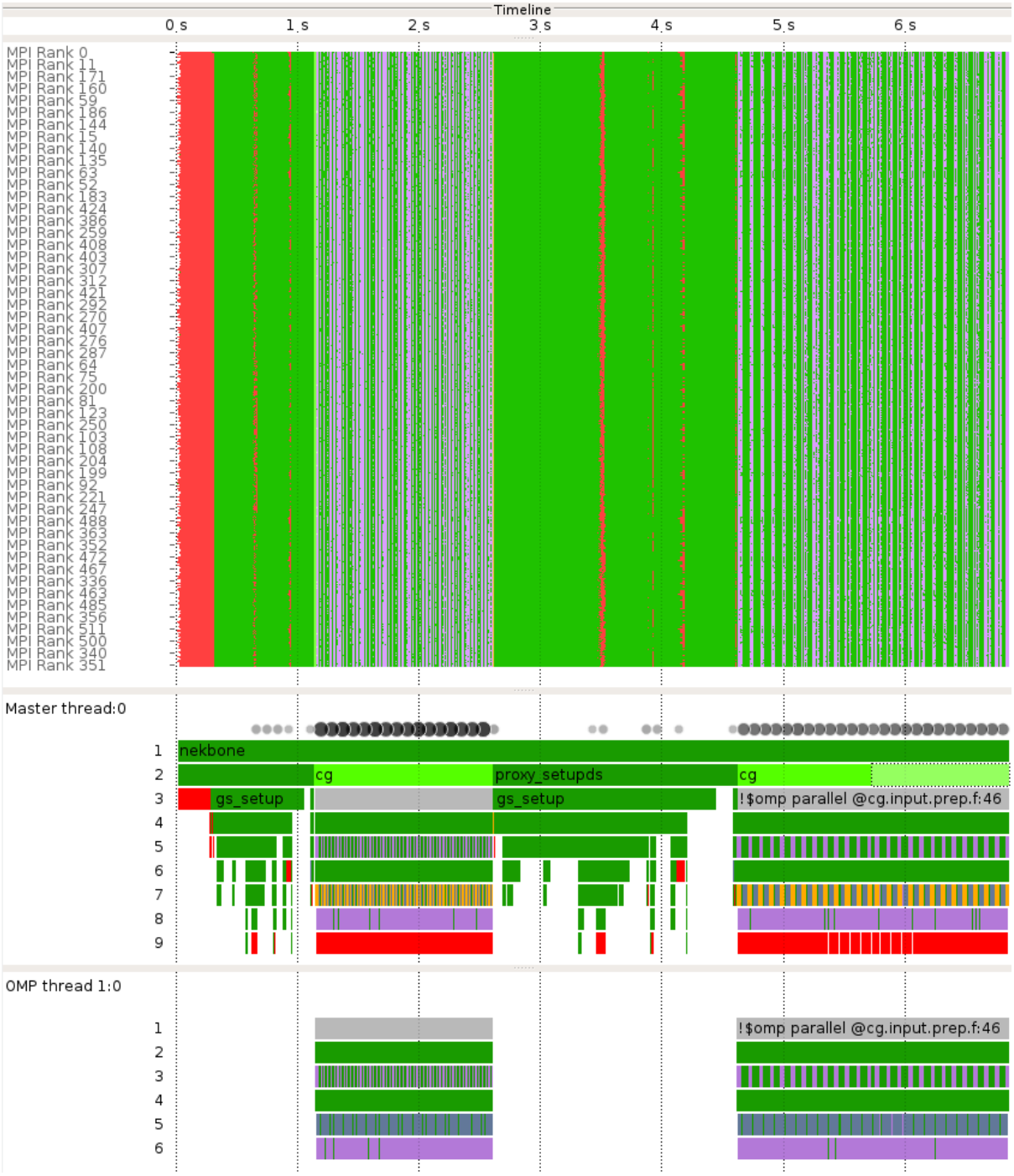

Figure 1: Execution timeline of Nekbone execution using 512 MPI processes each with 64 OpenMP threads on JUQUEEN.
The timeline state chart of all 512 MPI processes at the top (with only OpenMP master threads and worker threads not shown), followed by thread activity charts of MPI rank 0 Master thread and its OMP thread 1 at bottom showing two pairs of `cg` iterations (pale green), each preceded by `proxysetupds`. While the latter doesn't involve any threading, each `cg` iteration has 100 uniform steps employing OpenMP threading (lavender), `MPI_Allreduce` (orange) and MPI non-blocking point-to-point communication (red).

3 Application structure and Focus of Analysis

The timeline presentation of Nekbone execution in Figure 1[1] clearly distinguishes the two `cg` conjugate gradient phases where OpenMP threading is employed, each preceded by `proxy_setupds` which does not use threading. The first two `cg` instances have less computation (`nx1=9`) and are correspondingly faster than the second pair (`nx1=12`). Within the `cg` region chosen as focus of analysis (FOA) for auditting, regular MPI point-to-point and collective communication is performed only by the OpenMP Master thread of each MPI process.

(a) Visits (b) CPU Time [%]

Figure 2: Nekbone execution syntactic structure of `cg` FOA annotated with call-path visit counts and proportion of CPU time from execution with 28,672 MPI processes (1,835,008 threads) on JUQUEEN. (Routines with small execution times filtered.)

Figure 2 shows the `cg` extract of the Nekbone execution call-tree profile[2] from a measurement on JUQUEEN. The OpenMP `parallel` region is executed 4 times by each of the 1,835,008 threads, containing two significant parts. The `axi` routine executed 400 times by each thread dominates execution time, with MPI non-blocking point-to-point communication performed. On the other hand, the `glsc3i` routine contains MPI collective communication (`MPI_Allreduce`),

[1] Vampir display quick reference:
https://pop-coe.eu/sites/default/files/pop_files/vampir_display_quickref.pdf

[2] Cube display quick reference:
https://pop-coe.eu/sites/default/files/pop_files/cube_display_quickref.pdf

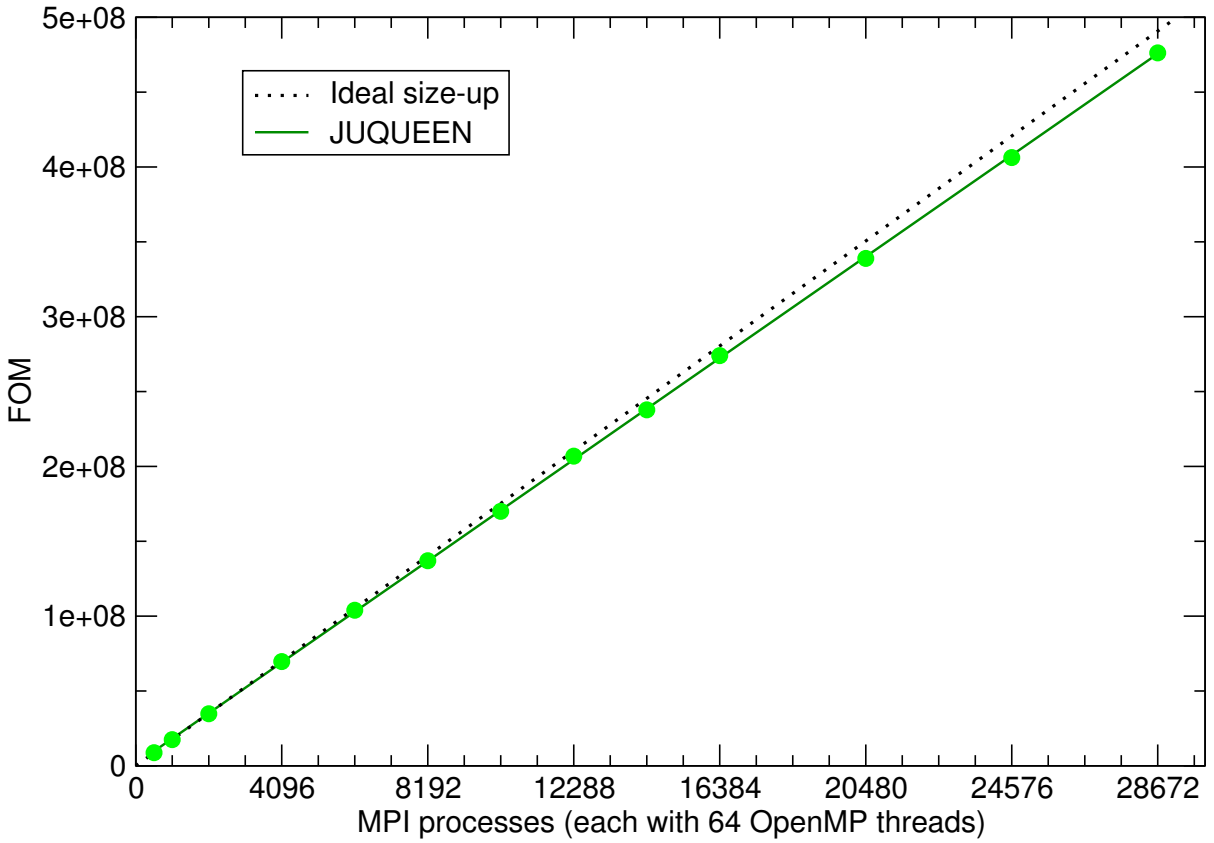

Figure 3: Weak scaling of Nekbone CORAL testcase figure-of-merit (FOM) on JUQUEEN with 64 OpenMP threads per MPI process on each compute node.

plus an expensive `critical` section. As MPI is used exclusively by Master threads (within OpenMP `master` blocks), `MPI_THREAD_FUNNELED` thread support level is appropriate.

4 Scalability

Figure 3 shows the weak scaling performance of the Nekbone CORAL testcase on JUQUEEN up to all 28,672 compute nodes. The reported figure-of-merit (FOM) from each execution is the average aggregate number of floating-point operations per second in the two CG phases. Scalability is seen to be very close to ideal and in line with the reference values provided by the CORAL benchmark guidance for BG/Q.

5 Parallel efficiency metrics

Basic parallel efficiency metrics[3] are defined where the higher the value (closer to 1.00) then the better is the efficiency. *Load balance efficiency* is the ratio of average computation to maximal computation time. *Communication efficiency* is the ratio of maximal computation to maximal executing time. *Parallel efficiency* is the ratio of the average computation time to the maximal executing time which is also the product of *Load balance* and *Communication*. *Computation*

[3]POP standard metrics for parallel performance analysis: https://pop-coe.eu/node/69

efficiency reflects how total computation time scales, here using the 512-process (half-rack, midplane) execution as reference.[4] *Instruction scaling* and *IPC scaling* similarly reflect how the number of computation instructions executed in total and executed per cycle (IPC) change with the number of MPI processes. *Global efficiency* is then the product of *Computation efficiency* and *Parallel efficiency*.

Table 1: Efficiency metrics for Nekbone `cg` FOA execution on JUQUEEN.

Racks	$\frac{1}{2}$	1	2	4	8	16	28
Processor distribution	8x8x8	16x8x8	16x16x8	16x16x16	32x16x16	32x32x16	32x32x28
Processes	512	1024	2048	4096	8192	16384	28672
Threads	32768	65536	131072	262144	524288	1048576	1835008
Global efficiency	**0.67**	**0.67**	**0.67**	**0.67**	**0.67**	**0.67**	**0.65**
- Parallel efficiency	**0.67**	**0.67**	**0.67**	**0.67**	**0.67**	**0.67**	**0.65**
- - Load balance efficiency	**0.96**	**0.96**	**0.96**	**0.96**	**0.96**	**0.96**	**0.95**
- - Communication efficiency	**0.70**	**0.70**	**0.70**	**0.70**	**0.70**	**0.70**	**0.69**
- - - Serialization efficiency	**0.70**	**0.70**	**0.70**	**0.70**	**0.70**	**0.70**	**0.69**
- - - Transfer efficiency	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
- Computation efficiency	*1.00*	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.97**
- - Instructions scaling	*1.00*	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
- - IPC scaling	*1.00*	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.98**

Table 1 gives an overview of the efficiency of the Nekbone weak-scaling testcase executions on JUQUEEN. It also reports the processor distribution determined by the benchmark execution for each number of processes. Efficiencies are the same at all scales, reflecting excellent weak scaling, with only a small degradation for the very largest execution configuration which has a non-power-of-2 z-processor distribution of 28. Computation efficiency is perfect with no inefficiency arising from additional work, and Load balance efficiency above 0.95 remains very good. Transfer efficiency is also perfect, whereas serialization efficiency of 0.70 reflects a constant amount of synchronization of neighbouring MPI processes and OpenMP threads, such that Parallel efficiency and Global efficiency are 0.67.

6 Load balance

Load balance efficiency is very good at over 0.95 for the FOA. As each thread is assigned an identical amount of computation, which is unchanged for configurations employing more threads when weak scaling, this is to be expected. The small variation by MPI rank relating to the problem decomposition and partitioning doesn't change with the number of processes employed. A small amount of additional work is evident on Master threads, which is where MPI communication is done.

7 Serial computation

Local computation comprises 54% of the total CPU time, with two-thirds in the `axi` routine executed in parallel by all OpenMP threads. Overall instructions executed per cycle for compu-

[4] Efficiency metrics are essentially unchanged for smaller machine partitions down to a single node-board with 32 processors.

tation is only 0.17 which is rather low, and only marginally better at 0.21 for `axi`, but largely explained by four OpenMP threads executing on a PowerPC A2 core (with separate hardware threads).

The computational kernel employs a large number of small dense matrix-matrix multiplications. Measurements with additional hardware counters could be used for more in-depth investigation in further analysis studies.

8 Communication & Synchronization

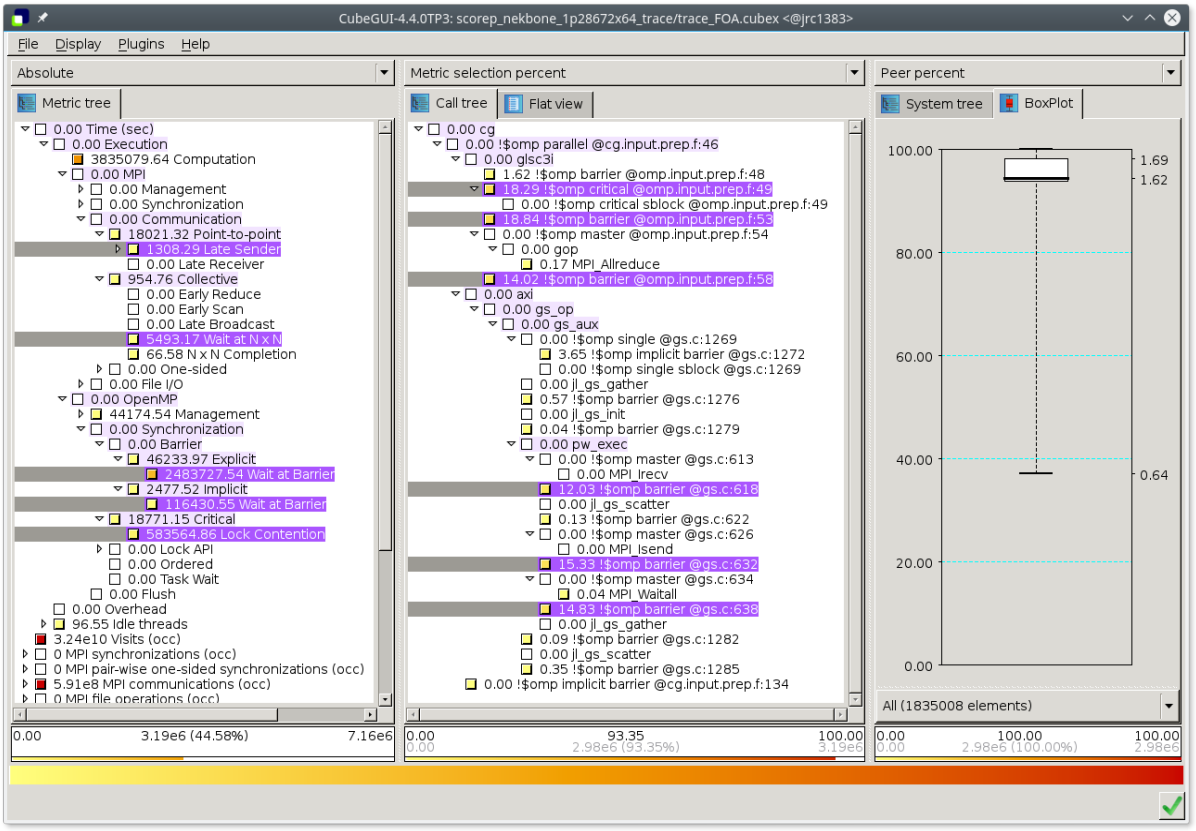

Figure 4: Scalasca trace analysis report explorer presentation of Nekbone `cg` extract from execution with 1,835,008 OpenMP threads on JUQUEEN. Wait states in MPI and OpenMP communication and synchronization selected from the Time metric hierarchy (left panel) amount to 45% of total CPU time, 93% of which is attributed roughly equally to five explicit OpenMP barrier synchronizations and a OpenMP critical section in `glsc3i` (middle panel). Variation by thread for these is summarized in the box-plot (right panel).

Communication efficiency of 0.70 is found to derive entirely from serialization or wait states at process and thread synchronizations. Only 0.4% of total CPU time is used for MPI (done by master threads of each process), and while most of this time is in point-to-point communication the majority of waiting time is for the `MPI_Allreduce` collective communication involving all processes. Non-blocking `MPI_Irecv` and `MPI_Isend` communication with all neighbours are completed with a single `MPI_Waitall`. OpenMP synchronization costs for all 64 threads amount to 45% of total CPU time, with the majority of this being for five explicit `barrier` synchronizations. Whereas implicit barrier synchronizations account for another 2%, lock contention

for the `critical` section in the `glsc3i` routine additionally contributes waiting time of over 8% of the total CPU time.

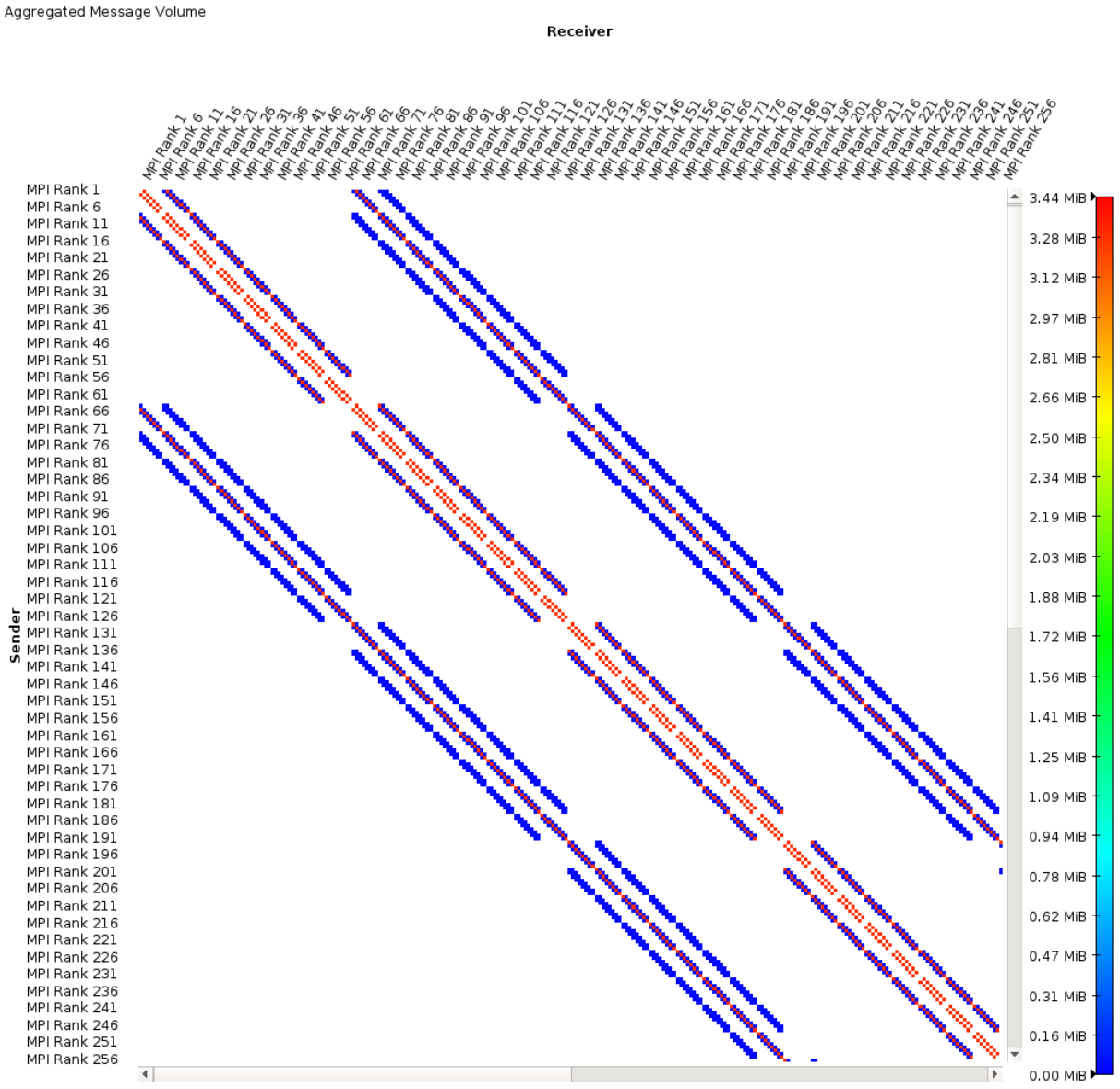

Figure 5: Point-to-point aggregate message volume communication matrix extract for `cg` FOA.

Figure 5 shows an extract of the MPI point-to-point communication matrix for one of the first `cg` instances, where over 3.3 MiB is exchanged in 100 messages with each of up to six neighbours and a further 54 kiB exchanged in 100 messages with up to a further twenty ranks in the vicinity. For the second `cg` instances, the message volumes are about twice as large, but with the same number of messages and communication partners.

9 File I/O

The only file operation is reading the small textual configuration file (`data.rea`, 336 bytes) during setup by MPI process 0, which is negligible.

10 Summary of observations

Scalability and parallel efficiency of Nekbone on the JUQUEEN Blue Gene/Q computer was investigated with a weak-scaling test configuration up to the full 28 racks (28,672 MPI processes, 1,835,008 OpenMP threads). The combination of MPI and OpenMP is able to make effective use of available memory and hardware threads on each compute node with 64 OpenMP threads per MPI process.

- Weak scaling performance of the benchmarked `cg` phase used as focus of analysis is very good, with only a minor degradation for the very largest execution configuration which has slightly lower efficiencies.
- Instructions scaling and Transfer efficiency are considered perfect, with very good Computation and Load balance efficiencies.
- Parallel efficiency of 0.67 arising from Serialization efficiency of 0.70 for all scales reflects constant numbers of global collective reductions, point-to-point communications with neighbour processes and synchronizations of OpenMP threads within processes.
- MPI non-blocking point-to-point communication is done efficiently and has a minimal impact on performance.
- The most significant wait states are associated with 5 explicit OpenMP `barrier` synchronizations and contention for a lock in an OpenMP `critical` section. Waiting time for the MPI collective `Allreduce` and point-to-point communication executed by OpenMP master threads are negligible in comparison.
- Use of all four hardware threads of each core benefits performance (and energy-efficiency) but results in lowered IPC (computational instructions executed per cycle) per thread.

Recommendations

- Investigate whether explicit OpenMP barriers are all necessary for correctness, as they constitute the most significant inefficiency.
- Investigate computation efficiencies of individual cores with hardware counter measurements, e.g., cache performance, vector instructions, branch instructions, etc. This should provide insight into the small dense matrix-matrix multiplication kernels.
- Compare performance and scalability with Cray computer systems, where the number of OpenMP threads per socket is likely to be lower and topological aspects of machine partitions (and other jobs) can be expected to influence performance.
- Compare performance of the matrix multiplication kernels using SIMD and/or LIBXSMM.

One or more of these recommendations could be pursued via follow-up POP services.