Supplementary File of the TPDS Manuscript

Juan-Antonio Rico-Gallego, Alexey L. Lastovetsky and Juan-Carlos Díaz-Martín

Abstract—This supplementary file contains the supporting materials of the TPDS manuscript "Model-Based Estimation of the Communication Cost of Hybrid Data-Parallel Applications on Heterogeneous Clusters". It clarifies some aspects which are briefly described or simply introduced in the paper due to limited space. We include the pseudo-code for the Wave2D kernel implementation, a deeper discussion on HLogGP and LMO models and a description of the τ–Lop tool, used in the main paper to automatize the evaluation of the τ–Lop analytical cost expressions of SUMMA and Wave2D kernels.

1 WAVE2D SOLVER

In addition to the Matrix Multiplication SUMMA algorithm used as a conduit example in the paper, we discuss here in detail the algorithm implemented for solving the 2D Wave equation

$$\frac{\partial^2 u}{\partial t^2} = c^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right). \quad (1)$$

The algorithm uses the technique of finite differences, ubiquitous in HPC, in a partitioned 2D mess. We use τ–Lop to model and estimate its communication cost. The Fig. 1 shows the discrete solution $u(x, y, t)$ of equation (1) at time $t = 102$, for particular initial and boundary conditions. In general, the function $u(x, y, t)$ is approached in the mesh $x \in (0, N)$, $y \in (0, N)$ and $t \in (0, T]$, with N the size in elements (double precision real numbers) of the mesh and T the number of considered iterations. In our implementation, along time, $u(x, y, t + 1)$ is given by successive instances of matrix New. New is generated from its previous instances, the matrices Cur and Old, $u(x, y, t)$ and $u(x, y, t - 1)$ respectively, according to the recursive finite differences algorithm driven by the following stencil, also shown at the right of Fig. 2:

$$\begin{aligned} New(i,j) =& 2(1-2C^2)Cur(i,j) - Old(i,j) \\ &+ C^2 Cur(i-1,j) + C^2 Cur(i+1,j) \\ &+ C^2 Cur(i,j-1) + C^2 Cur(i,j+1) \end{aligned} \quad (2)$$

Fig. 2 illustrates a 2D partition of the mesh between $P = 8$ processes running on two machines. The white rectangles are located at machine M_0 and the dark rectangles at machine M_1. Note that relation (2) imposes the communication of the perimeter (halo) elements between the Cur rectangles. This 2D partition depicts a scenario where a process has to communicate vertically and horizontally with several other processes. An alternative 1D partition of the data space in horizontal or vertical slices would lead to a more simple communication scheme, where each process has just two neighbor processes for interchanging data. In the figure, transmissions from p_1 to its neighbors

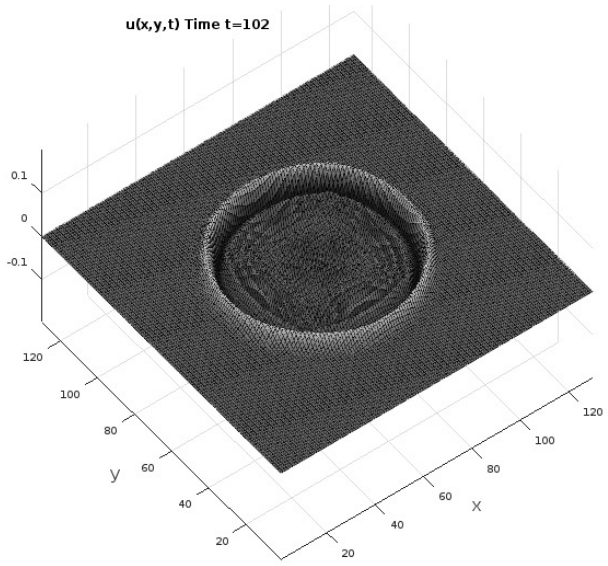

Fig. 1. Visualization of discrete solution $u(x, y, t)$ of equation (1) in a NxN data mesh with $N = 128$, at time $t = 102$, for particular initial and boundary conditions

are shown. For instance, if η_i denotes the neighborhood of p_i in the clockwise order, then $\eta_1 = \{2, 4, 3, 6, 0, 5\}$. Non-blocking point-to-point transmissions are used in the implementation. Assuming that all of them start at once, the τ-Lop cost per iteration allocated to p_i will be

$$\Theta_i = \mathop{\Vert}_{j \in \eta_i} T^{c(j)}(m(j)), \quad (3)$$

where $m(j)$ is the size of the message sent to neighbor p_j, and $c(j)$ is the channel used to send the message. As the transmissions of all P processes progress concurrently, the cost of each iteration is

$$\Theta = \left[\mathop{\Vert}_{i=0}^{P-1} \Theta_i \right] \quad (4)$$

The τ–Lop cost of the algorithm is hence $\Theta_{w2D} = T \times \Theta$. Algorithm 1 shows the pseudo-code of the Wave2D kernel[1].

We have pointed out that each process has to interchange the halos of the Cur matrix assigned rectangle with its

Juan-Antonio Rico-Gallego and Juan-Carlos Díaz-Martín are with the University of Extremadura, Avd. Universidad s/n, 10003, Cáceres, Spain. E-mail: {jarico,juancarl}@unex.es
Alexey L. Lastovetsky is with the University College Dublin, Belfield, Dublin 4, Ireland.E-mail: alexey.lastovetsky@ucd.ie

1. Full C code available at http://gim.unex.es/taulop.

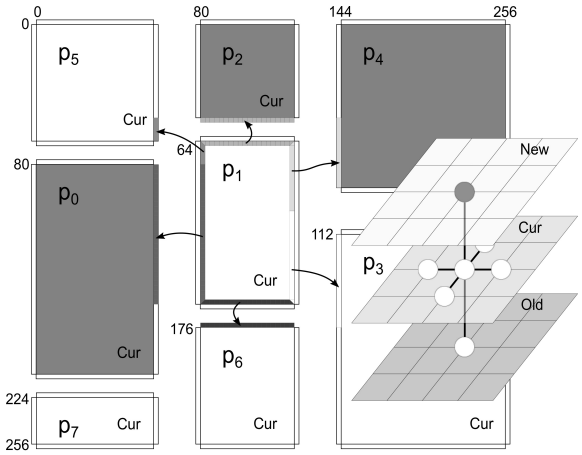

Fig. 2. A 2D partition for solving the wave equation. Each process recomputes its rectangles New, Cur and Old along time. The computation stencil imposes communications in Cur. Process p_1 sends its perimeter elements of Cur, which will form the halo of its neighbors.

neighbors. The function *getCurHaloCnt*(p_{src}, p_{dst}, cnt) returns the number of double precision elements two neighbor processes share. Its goal is to make the code independent of the data partition, either 1D or 2D. The function *getCurHaloAddr*$(p_{src}, p_{dst}, \uparrow halo\_addr)$ returns the address of the halo in the Cur matrix rectangle for sending it to a neighbor process. Fig. 2 shows that the partition scheme makes rectangles in the same column and with the same width. It is not necessary, hence, to pack and unpack the data in the *vertical* communication. However, data is packed in the send buffer $buff\_snd$ for communicating to a process in an adjacent column. As well, a process uses $Card(\eta)$ receive buffers for receiving data from its neighbors. Non-blocking communication is used for the halo interchange, so any reception (and send) is complete only after *MPI_Waitall* returns. It is now when the receiver, just in case, unpacks the received data. The communication time in the code is measured for each iteration, and it is finally calculated as $\sum_{t=1}^{T}(t_{end_t} - t_{start_t})$.

Algorithm 1 Measuring the cost of the Wave2D kernel

```
MPI_Comm_rank(WORLD, ↑me)
setInitialConditions(New, Cur, Old)
for p ∈ {η} do
    getCurHaloCnt(me, p, ↑cnt)
    buff_snd[p] = malloc(cnt × sizeof(double))
    buff_rcv[p] = malloc(cnt × sizeof(double))
end for

for t = 1 to T do
    // Computation
    wave2d_computation()

    // Communication
    MPI_Barrier(WORLD)
    n ← 0
    t_start ← MPI_Wtime()
    for p ∈ {η(me)} do
        getCurHaloCnt(me, p, ↑cnt)
        getCurHaloAddr(me, p, ↑halo_addr)

        if need_pack(me, p) then
            MPI_Type_vector(. . .)
            MPI_Pack(buff_snd[p] ← halo_addr, cnt)
        else
            buff_snd[p] ← halo_addr
        end if

        MPI_Isend(buff_snd[p], cnt, MPI_DOUBLE, p,
                  TAG, WORLD, ↑req[n++])
        MPI_Irecv(buff_rcv[p], cnt, MPI_DOUBLE, p,
                  TAG, WORLD, ↑req[n++])
    end for
    MPI_Waitall(n, req)
    for p ∈ {η(me)} do
        getCurHaloCnt(me, p, ↑cnt)
        getCurHaloAddr(me, p, ↑halo_addr)
        if need_unpack(me, p) then
            MPI_Unpack(halo_addr←buff_rcv[p], cnt)
        else
            halo_addr ← buff_rcv[p]
        end if
    end for
    t_end ← MPI_Wtime()
end for
```

2 Brief Description of Heterogeneous Models

Modern HPC platforms are composed of nodes of multicore CPUs and GPUs linked by channels with different performance and features. Proposals for analytically modeling heterogeneous platforms usually depart from established models and extend their functionality to cover such more complex scenarios. That is the case for *HLogGP*, based on *LogGP*, and *LMO*, based on the *Hockney* model. Both are point-to-point models which add parameters for dealing with the heterogeneity of the communication network and the processing elements. However, as *LogGP* and Hockney, *HLogGP* and *LMO* assume no contention in the communication. In particular, this lead to a poor expressive power of collective operations and often to significant inaccuracies of their cost predictions.

2.1 HLogGP

HLogGP [1, 2] is a model based on *LogGP* [3] that takes into account the heterogeneity of both the processor and the networks of the platform. *LogGP* characterized the communication cost according to the following parameters: the network latency (L), the overhead or time a processor is engaged in the transmission or reception of a message (o), the time interval between consecutive message transmissions (g), and the time interval between consecutive byte transmissions (G), which captures the network bandwidth for long messages. P was the number of processors in the cluster.

The scalar parameters of *LogGP* are expanded in *HLogGP* to represent the o_s, o_r (overhead in sender and receiver) and g as vectors of P components. L and G now depend on each pair of processors, so they become matrices of $P \times P$ components. In addition, the *HLogGP* model includes

the processor dependent computational power as a new parameter, denoted by P_i. It is defined as the amount of work per time unit a processor is able to finish. The cost of a point-to-point message transmission between processors p_i and p_j can be represented as:

$$T_{i\to j}(m) = L_{ij} + o_{S_i} + o_{R_j} + (m-1)\,G_{ij} \tag{5}$$

The heterogeneous platform used to validate the model is a small cluster of eight Pentium III nodes of two types, four nodes at 733 GHz and four nodes at 550 GHz. These nodes are connected by two different Ethernet networks, of 10 and 100 MBit/s respectively. The model is used to predict the execution time of a volumetric magnetic resonance image compression application. Bosque et al. [2] provide with a methodology to measure the *HLogGP* parameters based on self-made benchmarks. Note that the parameters have to be measured for each pair of processors in the system, and hence, the number of tests is of order $O(P^2)$, a figure that keeps manageable for this small system. The sending overhead o_S is measured by evaluating the response time of a blocking send MPI function, using short message sizes. No measured technique is described for the receiving overhead o_R, and hence, one have to suppose that it is taken as the sending counterpart. Latency L is measured as the time to send a point-to-point message of size 1 byte. The gap per message g is measured by sending consecutive messages and evaluating the send function response time. The gap per byte G is measured as the time of sending large messages, subtracting the sender overhead and latency. Finally, the computational power P_i is estimated in each node as the inverse of the computational time of a benchmark.

Following, we model the Wave2D kernel taking as a base the expression (5) for each point-to-point communication. All the transmissions are assumed to start at once. FuPerMod is used to balance the computational load illustrated by Fig. 2. Anyway, as Code 1 shows, we use a *barrier* at the beginning of the communication phase. The cost per iteration allocated to a process p_i will be:

$$\Theta_i = \sum_{j\in\eta_i} T_{i\to j}^{c(j)}(m) \tag{6}$$

where m is the size of the message sent to neighbor p_j, and $c(j)$ is the channel used to send the message. The transmissions of all P processes progress concurrently, however, *HLogGP* does not provide any mechanism to model concurrent transmissions, and it models the cost of each iteration as the maximum of the processes cost according to:

$$\Theta = \max_{i=0}^{P-1} \Theta_i \tag{7}$$

The *HLogGP* Wave2D kernel communication cost is finally calculated as $\Theta_{w2D} = T \times \Theta$.

2.2 LMO

Lastovetsky et al. [4] address hierarchical communications on an Ethernet network connecting a set of heterogeneous processors. The model proposed, *LMO*, targets the impact of the heterogeneity of the processors on the communication cost of a set of operations, namely point-to-point, one-to-many (scatter and gather) and broadcasting. *LMO* carefully separates the cost related to the processors and the network for the sake of more accurate communication cost predictions. The model assumes no contention in the communication channel. Being based on the Hockney model [5], *LMO* is presumed to be less accurate than *HLogGP*. The cost of a message transmission between the processors p_i and p_j is

$$T_{i\to j}(m) = L_{i\,j} + C_i + m\,t_i + C_j + m\,t_j + \frac{m}{\beta_{ij}}, \tag{8}$$

where m represents the message size in bytes. C_i is the *fixed processing delay* of process p_i, and reflects the heterogeneity of the processors. t_i is the *per-byte delay*, and reflects the heterogeneity of communication channels. The *fixed network delay* parameter $L_{i\,j}$ improves the flexibility to express the execution time of collectives, and is added to the original model by Lastovetsky and Richkov in [6]. β_{ij} is the transmission rate of the channel connecting the processors p_i and p_j.

LMO and its parameter measurement procedure are deeply discussed by Lastovetsky et al. in [7] and [8]. Authors describe a comprehensive experimental methodology to build their *LMO* model on a 16-node heterogeneous cluster. Like *HLogGP*, in a generalized P-node cluster the number of parameters is of order $O(P^2)$, specifically $2P + P^2$, that is, P parameters C_i, P parameters t_i, and P^2 transmission rates $\beta_{i\,j}$ between each pair of processors i and j. As point-to-point communication experiments do not provide sufficient data for the estimation of so many parameters, some particular collective experiments between triplets of processors are performed. Then, a linear system of equations $Ax = b$ with the parameters as unknowns and the execution times of the communication experiments as a right hand side is built and solved.

3 An Introduction to the τ–*Lop* Tool

The τ–Lop Tool is a software library for automating the evaluation of τ–Lop expressions. Such expressions are the analytical representation of the communications cost of data parallel algorithms, running in either homogeneous or heterogeneous platforms. The expression (4), which accounts for the cost of our 2D wave kernel, is just an example.

Next follows a brief introduction to the τ–Lop Library, including some use cases[2]. The principle underlying the τ–Lop library is that the τ–Lop cost can always be represented as a *set of sequences of transmissions progressing concurrently*, for instance $\big(T^1(m_a) + T^0(m_b) + T^1(m_b)\big) \parallel \big(T^0(m_b) + T^1(m_a)\big) \parallel \big(T^1(m_a) + T^0(m_b)\big)$. The key ability of the library is storing the expression as a data structure illustrated by Fig. 3. Note how the operative data structure reflects the formal structure of the expression, namely, three sequences of point-to-point transmissions that take place in parallel. Of course, the library provides a procedural interface to build these (abstract) data structures. In fact, the library provides three main interfaces.

The cost expressions involve transmissions $T(m)$. To be evaluated, a transmission must be expanded to an expression of just transfer times L and overheads o. For instance $2\parallel T^0_{p2p}(m)$ may expand to $2\parallel \big[o^0(m) + 2L^0(m,1)\big] =$

2. The τ–Lop tool is available from http://gim.unex.es/taulop.

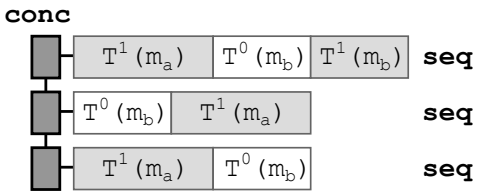

Fig. 3. Internal representation in the τ–Lop library of the cost expression $\left(T^1(m_a) + T^0(m_b) + T^1(m_b)\right) \parallel \left(T^0(m_b) + T^1(m_a)\right) \parallel \left(T^1(m_a) + T^0(m_b)\right)$ as a set of three concurrent sequences of transmissions. The transmissions have different message sizes and progress through different channels, plot with different background color. Time goes from left to right.

$o^0(m) + 2L^0(m, 2)$. One has to instantiate these expanded expressions with the values that $L(m, \tau)$ and $o(m)$ take in the platform. The library hence provides a second interface to set the parameters of the model in the working platform. Finally a third interface triggers the evaluation of an expression.

3.1 Modeling Simple Transmissions

The fundamental software object of the τ–Lop Library is the *Transmission*, which represents an individual τ–Lop transmission cost of the form $\tau \parallel T^c(m)$, with m the size of the message and τ the number of messages progressing concurrently through the channel c. Code 2 shows the creation of a Transmission object.

Code 2 A simple transmission.

```
Transmission *T = new Transmission(c, m, τ);
```

Besides the Transmission, the library provides other objects. One of them is the *Process* object, which has two attributes, the node where it runs and its rank in the algorithm. Code 3 shows that there is an alternative way of creating a Transmission, namely, by specifying the source and destination processes instead of the channel. The channel is internally determined by the processes location.

Code 3 A simple transmission between two processes.

```
Process *src = new Process ( src_rank, src_node);
Process *dst = new Process (dst_rank, dst_node);
Transmission *T = new Transmission(src, dst, m, τ);
```

As a τ–Lop cost expression is a set of sequences of transmissions progressing concurrently (Fig. 3), the τ–Lop library adopts a composite model. It provides with two new objects: *TauLopSequence* and *TauLopConcurrent*. *TauLopSequence* contains one or more transmissions carried out in sequence. *TauLopConcurrent* contains a set of *TauLopSequence* objects that progress in parallel. Code 4 creates a sequence of two transmissions through the same channel c of the form $T^c(m_1) + T^c(m_2)$. Meanwhile, the Code 5 creates two concurrent transmissions on the same channel c, represented in τ–Lop as $T^c(m_1) \parallel T^c(m_2)$.

Concurrent transmissions with different message sizes appear in more heterogeneous scenarios. Also transmissions progressing through different channels. Code 6 implements the expression of Fig. 3.

Code 4 A sequence of two transmission.

```
TauLopConcurrent *conc = new TauLopConcurrent ();
TauLopSequence *seq = new TauLopSequence ();
seq→add(new Transmission(c, m1, 1));
seq→add(new Transmission(c, m2, 1));
conc→add(seq);
```

Code 5 Two concurrent transmission.

```
TauLopConcurrent *conc = new TauLopConcurrent ();
TauLopSequence *seq;
seq = new TauLopSequence ();
seq→add(new Transmission(c, m1, 1));
conc→add(seq);
seq = new TauLopSequence ();
seq→add(new Transmission(c, m2, 1));
conc→add(seq);
```

Code 6 Three concurrent sequences.

```
TauLopConcurrent *conc = new TauLopConcurrent ();
TauLopSequence *seq;

seq = new TauLopSequence ();
seq→add(new Transmission(1, ma, 1));
seq→add(new Transmission(0, mb, 1));
seq→add(new Transmission(1, mb, 1));
conc→add(seq);

seq = new TauLopSequence ();
seq→add(new Transmission(0, mb, 1));
seq→add(new Transmission(1, ma, 1));
conc→add(seq);

seq = new TauLopSequence ();
seq→add(new Transmission(1, ma, 1));
seq→add(new Transmission(0, mb, 1));
conc→add(seq);
```

The *TauLopCost* object contains the evaluation of a *TauLopConcurrent* object. Code 7 shows an example.

Code 7 Calculating the communication time of a *TauLopConcurrent *conc* expression.

```
TauLopCost *tc = new TauLopCost();
conc→apply(tc);
double t = tc→getTime();
```

3.2 Modeling Advanced Transmissions

The τ–Lop Library provides with an extensible interface for modeling and estimating the cost of MPI-like collective operations, implemented using different algorithms. A collective executes in the context of a *communicator*, a central concept to MPI and defined in the τ–Lop Library as a group of processes with an associated mapping. Code 8 builds a *broadcast* operation implemented as a binomial tree, and predicts its cost given a communicator of P processes deployed in sequential mapping on a homogeneous multicore machine with P/Q nodes, being Q the number of cores per node.

Code 8 Calculating the communication time of a collective operation.

```
Communicator *world = new Communicator (P);
Mapping *map = new Mapping(P, Q, MAPPING_SEQ);
world→map(map);
Collective *bcast = new BcastBinomial();
double t = bcast→execute(world, &m, root);
```

Nodes in a machine are identified as integer numbers in the range $[0, M-1]$. In a communicator, processes are identified by their ranks from 0 to $P-1$. The *Sequential mapping*, known in the library as *MAPPING_SEQ*, assigns a process rank to a node as $M_{rank} = P_{rank}/M$. Other commonly used mappings, and even irregular mappings can be provided.

3.3 Modeling a Full Kernel

Following, the communication modeling and cost prediction code for the Wave2D kernel is discussed. Code 9 implements the expressions (3) and (4).

Code 9 Communication cost of the Wave2D kernel.

```
double wave2d (Communicator *comm) {
  double t = 0.0;
  int P = comm→getSize();
  TauLopConcurrent *conc = new TauLopConcurrent();
  for (int src = 0; src < P; src++) {
    for (int dst = 0; dst < P; dst++) {
      int size = getBoundary(src, dst);
      if (size > 0) {
        TauLopSequence *seq = new TauLopSequence();
        int c = (comm→getNode(src) ==
                 comm→getNode(dst)) ? 0 : 1;
        int m = size * sizeof(double);
        seq→add(new Transmission(c, m, 1));
        conc→add(seq);
      }
    }
  }
  TauLopCost *tc = new TauLopCost();
  conc→apply(tc);
  t = tc→getTime();
  delete tc;
  delete conc;
  return t;
}
```

The algorithm takes as a parameter the communicator defining the mapping of the processes in the platform, and hence the communication channel used by any pair of processes. The function *getBoundary(P_{src}, P_{dst})* returns the number of halo elements (double precision values) of each two processes, if any. Each transmission of a process to its neighbors is created and added to a *TauLopSequence* object. Each *TauLopSequence* structure then compose a concurrent *TauLopConcurrent* object, which will be finally evaluate for predicting the cost of the kernel.

REFERENCES

[1] J. Bosque and L. Perez, "HLogGP: a new parallel computational model for heterogeneous clusters," in *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, April 2004, pp. 403–410.

[2] J. L. Bosque and L. Pastor, "A parallel computational model for heterogeneous clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, pp. 1390–1400, 2006.

[3] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, "LogGP: incorporating long messages into the LogP model - one step closer towards a realistic model for parallel computation," in *Proc. of the seventh annual ACM symposium on Parallel algorithms and architectures*, ser. SPAA '95, NY, USA, 1995, pp. 95–105.

[4] A. Lastovetsky, I.-H. Mkwawa, and M. O'Flynn, "An accurate communication model of a heterogeneous cluster based on a switch-enabled Ethernet network," in *Parallel and Distributed Systems, 2006. ICPADS 2006. 12th International Conference on*, vol. 2, 2006, pp. 6 pp.–.

[5] R. W. Hockney, "The communication challenge for MPP: Intel Paragon and Meiko CS-2," *Parallel Comput.*, vol. 20, no. 3, pp. 389–398, Mar. 1994.

[6] A. Lastovetsky, V. Rychkov, and M. O'Flynn, "Revisiting communication performance models for computational clusters," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, 2009, pp. 1–11.

[7] A. Lastovetsky and V. Rychkov, "Building the communication performance model of heterogeneous clusters based on a switched network," in *Cluster Computing, 2007 IEEE International Conference on*, Sept 2007, pp. 568–575.

[8] A. Lastovetsky and V. Ryhkov, "Accurate and efficient estimation of parameters of heterogeneous communication performance models," *Int. J. High Perform. Comput. Appl.*, vol. 23, no. 2, pp. 123–139, 2009.