Statically Derived Data Access Patterns for NUMA Architectures *

Gianluca Turin, Einar Broch Johnsen, and S. Lizeth Tapia Tarifa

Department of Informatics, University of Oslo, Norway
{gianlutu,einarj,sltarifa}@ifi.uio.no

Abstract

A parallel program can be thought of as a collection of communicating tasks. When executing such a parallel program in a distributed setting, remote data access may introduce significant communication latency. Network contention is the main factor affecting latency. Therefore, it is important to reduce the amount of remote data access, in terms of distance travelled. Network contention can be significantly affected by the long distances covered while accessing remote data in architectures such as large mesh-network or torus-network machines. For example, BlueGene/P is a huge torus topology which may have as many as 130k processing elements. In particular, it is important to reduce data movement on NUMA machines based on such topologies. Several techniques have been developed for this purpose; for example, there are different heuristics to efficiently schedule tasks given their dependency graph.

In this work, we consider a static analysis of data locality in programs to reduce latency of hardware communication by means of topology-aware task scheduling, and thus to improve execution performance on large high performance computing systems. Our analysis enables the automatic extraction of dependency graphs, instead of manually specifying the communication pattern of an application or running it in a traced execution to collect this information. Our analysis is formalised as a type and effects system, we here describe the basic idea without explaining the details of the formalisation.

1 Motivation

The main resources in a large parallel machine are its cores and the network infrastructure. Optimal resources usage is key to tighten the gap between theoretical peak performance and actual peak performance. For example, IBM BlueGene/P [6] features flat 3D torus networks where each node connects directly to its six neighbours. Unless each node only communicates with its physically nearest neighbours, nodes have to share network links with other communication. Unless the physical placement of application processes matches the communication characteristics of the application, such sharing can result in significant communication contention and performance loss [3].

In theory any mapping of processes to cores is possible, in practice different systems have different restrictions on what mappings they allow. For example, IBM Blue Gene/P allows different combinations of process mappings along the X, Y, Z dimensions of the 3D torus, and BG/P systems consist of four cores per compute node, which can be considered to be a fourth dimension represented as T. An application can be mapped using different mappings, such as $TXYZ$, $XYZT$, and $TYXZ$. Other mappings that do not form a symmetric ordering of ranks in one of these orders are not supported. Thus, the "best performance" we can achieve is artificially restricted by this requirement [3].

* *ADAPt: Exploiting Abstract Data-Access Patterns for Better Data Locality in Parallel Processing* (www.mn.uio.no/ifi/english/research/projects/adapt/).

In most supercomputers, tasks are scheduled statically but they can migrate from the queue of one core to another due to job stealing or automatic workload balancing strategies. These strategies generally improve performance, but they may increase distances and total latency during execution.

2 A static analysis for data access patterns

To formalise information about locality in a fine-grained way on NUMA architectures, we propose a static analysis expressed using types and effects (e.g., [2]), which combines types with *locations* for a small core language targeting parallel hardware, where each assignment, function call or expression evaluation may involve long distance remote data access. In this language, new locations can be dynamically created and new tasks dynamically spawned. A task scheduling strategy decides on which core the tasks will execute. Our notion of locality is inspired by the Partitioned Global Address Space model (PGAS) [8]. This programming model provides a memory abstraction on parallel hardware that considers the memory partitioned in places: each task can only access its local memory directly and a global shared space is used across the computing nodes to share data among tasks.

Since the distances covered by remote accesses can affect the performance significantly on large networks [4], our goal is to investigate whether making information about the data access patterns of the different tasks available to the task scheduler can improve the overall performance. For this purpose, information of the network topology needs to be provided to the static analyser. A formal description of a given network with a metric capturing distances, can be obtained by letting the metric between each pair of processors reflect, e.g., the number of hops, the average or the expected latency of a packet. A metric based on hop-bytes (hops per byte) can approximate the overall contention on the network [1, 5]. Although the metric does not capture hot-spots created on specific links, it is easily derivable and correlates well with the actual application performance when communication to computation ratio is high [4].

Consider as a simple example an assignment instruction $x = y$, where x and y are at different locations in memory. These locations can be captured in a type system such that x has a location type of the form $L_a \cdot T$ where the location L_a abstractly describes the memory location that must be accessed to write the variable x on the left hand side of the assignment, and T describes the (standard) type of the value that can be written. Similarly for y the type may look like $L_b \cdot T'$. Again, the element L_b indicates a location in memory that we need to read to obtain the value of y and the T' indicates the type of that value.

Example: computing the average of distributed data

Let us consider a task for computing the average of a large distributed array: a set of processes have to locally reduce their array partition to the sum of all the elements. Then all those values have to be sent to a master process that will compute and print the total average.

Many parallel languages and libraries would be suitable to code this specification on a NUMA machine, by relying on message passing to achieve synchronisation. Nevertheless the recent trend has been to develop languages and libraries for NUMA machines providing the same programmability of standard programming languages [7].

In our core language such code could be implemented as in Figure 1. In this example it is possible to estimate the cost of the assignment of the variables in the array p_sums by checking the locality information coded in their location types.

```
task<L_T> worker(sums:L·L_T · ℕ){
    local L_p_sum, L, L_i, L_a1, ..., L_a25;
    var id: L·ℕ;
    var p_sum:L_p_sum · ℕ;
    var a[25]: L_a1...a25 · ℕ;
    var i: L_i · ℕ;

    for(i=0; i<25; i++) { a[i] = random(); p_sum = p_sum + a[i]; };

    *sums = p_sum; }
{
    local L_i, L_p_sums1, ..., L_p_sums4, L_avg;
    var p_sums[4]: L_p_sums1...p_sums4 · ℕ; // initially null
    var avg: L_avg · ℕ; // initially null
    var i: L_i · ℕ;

    for(i=1; i<5; i++) { spawn <L_p_sums_i>worker(&p_sums[i]); };

    for(i=1; i<5; i++) { while(p_sums[i]!=null); }; // active wait for p_sums

    for(i=0; i<5; i++) { avg = avg + p_sums[i]; };
    avg = avg / 100;
    print avg; }
```

Figure 1: Example: location types for a task computing the average of a large distributed array.

Given a sequence of five contiguous cores, the optimal scheduling is to place the master process, calculating the average, in the middle. Such configuration will give also the minimal total cost in our analysis.

While the distance covered by four integer values is far away to affect performances, a common scenario in parallel programming can be having streams of data converging to a filtering process. If few streams are enough to saturate the capacity of the network, more streams would require another filtering process. A scheduling able to avoid intersection of streams in this case would reduce network contention and thus latency.

Discussion and future work

When analysing a statement, we can use location types to estimate the cost of accesing data at different locations, according to the metric of the topology. This fine level of granularity assumes that a statement can remotely read and write variables, such that dependencies are evaluated on an instruction basis. The analysis then collects the derived costs according to the control flow; e.g., accumulating the cost of the two statements for sequential composition and taking the worst case for conditionals, and iterating over a for-loop. The granularity of the analysis and thereby the complexity of leveraging the analysis results, can be reduced as necessary by clustering locations to obtain a simpler dependency graph from the more complex instructions dependency graph. This way, the derived data access patterns can be simplified enough to be exploited by a scheduling strategy with acceptable overhead.

Since the analyser knows the topology metric, it can calculate for each task the possible cost of executing that task on the different cores. At runtime, the task scheduler can combine these statically derived data access patterns with runtime information in order to decide on the placement of a task. Our work so far is thus focused on how to statically derive data access patterns, in future work we plan to investigate how to extract these patterns automatically and to explore how they can be efficiently combined with runtime information by a task scheduling

strategy, to improve data movement overhead on huge NUMA machines.

References

[1] T. Agarwal, A. Sharma, A. Laxmikant, and L. V. Kale. Topology-aware task mapping for reducing communication contention on large parallel machines. In *Proc. of 20th Int. Parallel Distributed Processing Symposium, IPDPS 2006*. IEEE, 2006.

[2] T. Amtoft, H. R. Nielson, and F. Nielson. *Type and Effect Systems: Behaviours for Concurrency*. Imperial College Press, 1999.

[3] P. Balaji, R. Gupta, A. Vishnu, and P. Beckman. Mapping Communication Layouts to Network Hardware Characteristics on Massive-scale Blue Gene Systems. *Comput. Sci. Res. Dev.*, 26(3–4):247–256, 2011.

[4] A. Bhatele. *Automating Topology Aware Mapping for Supercomputers*. PhD thesis, University of Illinois at Urbana-Champaign, 2010.

[5] A. Bhatele and L. V. Kale. Heuristic-based techniques for mapping irregular communication graphs to mesh topologies. In *Proc. of 13th IEEE Int. Conf. on High Performance Computing and Communications, HPCC 2011*, pages 765–771. IEEE, 2011.

[6] Overview of the IBM Blue Gene/P project. *IBM J. Res. Dev.*, 52(1.2):199–220, 2008.

[7] J. Kepner. HPC productivity: An overarching view. *Int. J. High Performance Comput. Appl.*, 18(4):393–397, 2004.

[8] V. Saraswat, G. Almasi, G. Bikshandi, C. Cascaval, D. Cunningham, D. Grove, S. Kodali, I. Peshansky, and O. Tardieu. The asynchronous partitioned global address space model. In *Proc. of 1st ACM SIGPLAN Workshop on Advances in Message Passing*. ACM, 2010.