reading notes for supercomputing journal value 74 no. 4

zwpdbh

April 11, 2018

Contents

1 Parallel implementations of the 3D fast wavelet transform on a Raspberry Pi 2 cluster [2]

Present and evaluate 3 parallelization strategies of the 3d fast wavelet transform on a cluster of Raspberry Pi 2 SDCs. Using booth pthread and MPI.

1.1 Introduction

- In this work, they study the parallelization of the 3d fast wavelet transform(3D-FWT) on a cluster of raspberry pi 2 SDC(single-board computer)s. Using pthread and MPI for evaluating 3 different parallelization strategies on a cluster of 4 Pis.
- Pthread version are restricted to runs on a single boards, and MPI version can span to several Pis.
- Performance drops when all MPI processes spread to several boards due to the the limited bandwidth of the onboard LAN port.
- Overall, they have shown Raspberry Pi 2 SDC reveals as an appealing platform for giving support to 3D-FWT-based applications with low-cost and energy efficiency requirements.

2 A mathematical model to calculate real cost/performance in software distributed shared memory on computing environments [8]

2.1 Introduction

- cost/performance ratio is an important factors in HPC(high-performance computing), which acts as an economic justification for running scientific program on HPC system.
- This ratio parameter specifies the type of scientific program that can run in HPC systems like Cluster, Grid, and Peer-to-Peer (P2P) [12].
- cost calculate of a system includes: the execution of applications in HPC system, the mechanisms used to calculate the cost(two method).
  1. The first solution is the mathematical model of calculating cost and efficiency of each scientific program or computing system management application or any special feature of the computing system.

     The most important feature of the minor fee calculation is the process of using program execution for the calculation of cost and the cost/performance ratio. The most important advantage of this method is the feature of proposing the exact cost of a program execution in certain computing systems.

2. The second solution is using a public pattern to calculate the cost and mentioned coefficient in computing systems.

- One of the most important questions that should be answered while calculating the inter-process cost in computing systems is the cost calculation pattern.
- IPC (inter-process communication)
- DSM (distributed shared memory)

2.2 Related work

- SMP (symmetric multi-parallel)

2.2.1 Review of distributed shared memory based on system approach

2.2.2 General parameters of cost in DSM system based on system approach

(. . . boring topic for now)

3 An efficient anonymous authentication protocol in multiple server communication networks (EAAM) [5]

Propose an efficient anonymous authentication protocal in multiple server communication networks, called EAAM protocal, which is able to establish user anonymity, mutual authentication, and resistance against know security attacks.

THe novelty of the proposed scheme is that it does not require a secure channel during the registration between the user and the registration center and is resistant to a curious but honest registration system.

3.1 Conclusion

The main novelty of the protocol is that it provides resistance against a honest-but-curious RC(registration center).

(not very relavent to parallel computing)

4 Strategy for data-flow synchronizations in stencil parallel computations on multi-/manycore systems [11]

An innovative strategy for the data-flow synchronization in shared-memory system is proposed. This trategy assumes to synchronize only interdependent threads instead of using the barrier approach. Their proposed approache is evaluated for various Intel microarchitectures and done comparision with OpenMP barrier. It show it is better for 1.3 times.

4.1 Introduction

The main idea of this strategy is to synchronize only interdependent threads instead of using the barrier approach that—in contrast to our approach—synchronize all threads. An inseparable part of this strategy is the scheme of thread interrelationships for a given application. In fact, the data dependencies, workload distribution, way of parallelization and inter-thread data traffic play a key role in the effective adaptation of this strategy to a given application.

- the state-of-the-art synchronization alogrithms differ in trade-off between communication complexity, length of the critical path and memory footprint [5].
- The barriers are an essential sychronization approach for parallel models of many shared-memory programming languages such as OpenMP, OpenCL or Cilk. They can be grouped into three categories: centralized, tree and butterfly.
- Each synchronization algorithm features its own set of trade-off, where the areal profit is largely dependent on the structure of a computing system.

The main aim of this work is to avoid global barriers: the synchronization process should proceed only between carefully selected threads that depend on each others. An justification and study of related work meeting this challenge can be found in work [3].

- what is dynamic task graph?
- An other synchronization strategie: Data-flow communication layers are very popular in distributed-memory programming standards, including MPI or hStreams programming library

- In both cases, the synchronization between the interdepen- dent processing elements is explicitly defined according to communication flows of data, using the specific commands such as MPI_{Send} and MPI_{Recv} in the case of MPI.

So, there are two strategies for synchronizations:

1. barrier based, used mainly in shared-memory model
2. data flow based, used mainly in distributed-memory model

The author use data flow based, in shared-memory model.

5 Language-based vectorization and parallelization using intrinsics, OpenMP, TBB and Cilk Plus [?]

This paper evaluate OpenMP, TBB and Cilk Plus as basic language-based tools for simple and efficient parallelization of recursively defined computatinal problems and other problems that need both task and data parallelization techniques.

Show how to use these models to utilize multiple cores of modern processes.

- tuning data structures for better utilization of vector extensions of modern processors.
- Manual vectorization techniques based on Cilk
- Intel SIMD Data Layout Template containers

5.1 Introduction

Intel C/C++ compilers and development tools offer many language-based extensions that can be used to simplify the process of developing high-performance parallel programs.

- OpenMP
- Threading Building BLocks (TBB)
- Cilk Plus
- *intrinsics*, which all to utilize Intel Advanced Vector Extensions explicitly [?].

- SDLT template library can be applied to introduce SIMD-friendly memory layout transparently [?].

5.2 Short overview of selected language-based tools

- OpenMP
- TBB is a C+++ template library supporting task parallelism on Intel multicore platformcs.
- Cilk Plus adds simple language extensions to the C and C++languages to express task and data parallelism.
- Intrinsics for SIMD instructions allow to take full advantage of Intel Advanced Vector Extensions what cannot always be easily achieved due to limitations of programming languages and compilers. They all programmers to write constructs that look like C/C++ functions calls corresponding to actual SIMD instructions.
- SDLT (SIMD Data Layout Template) is a C++11 template library which provides containers with SIMD-friendly data layouts.

5.3 Conclusion

- They compare the speedup of the different implementation against sequential version of code. Depend on whether it is data parallel or task parallel, the performance varys between those librarie

6 Parallelization of stochastic bounds for Markov chains on multicore and manycore platforms [6]

Demonstrates the methodology for parallelizing of finding stochastic bounds for Markov chains on multicore and manycore platformcs

- involve a lot of irregular memory access.
- using OpenMP(for loop parallelism) and Cilk Plus (for task-based parallelism)
- compare the execution time and scalability

6.1 Experimental results

- time
- speedup

6.2 Conclusion

This paper presents the strength of the OpenMP standard for parallelizing with the use of `#pragma omp parallel for` which is data parallelism in OpenMP.

6.3 Things to do:

need to state clearly what is the main differences between task parallelism and loop (data) parallelism.

7 A taxonomy of task-based parallel programming technologies for high-performance computing [13]

Provide an initial task-focused taxonomy for HPC technologies, which covers programming interface and runtime mechanisms.

7.1 Introduction

- In HPC domain, loop-based and message-passing paradigms are dominant. We specifically aim to categorize task-based parallelism technologies which are in use in HPC
- Definition of task: a sequence of instructions within a program that can be processed concurrently with other tasks in the same program.
- Several languages are common in HPC domain:
  1. Cilk
  2. OpenMP
  3. TBB
  4. Qthreads
  5. Argobots
  6. StarGPU

7. Chapel
8. X10
9. HPX
10. Charm++

- Each task-based environment has two central components:
    1. programming interface
    2. runtime system

8 Hybridworkstealingoflocality-flexibleandcancelabletasksfortheAPG library [10]

8.1 Abstract

- parallel programs should be albe to deal with both shared memory and distributed memory
- propose a hybrid work stealing scheme, which combines the lifeline-based variant of distributed task pools with the node-internal load balancing of Java's Fork/Join framework.
- APGAS library for Java, which is a branch of the X10 project.

9 Pythonacceleratorsforhigh-performancecomputing [9]

9.1 Abstract

- python is popular and is slow; python community drive the effort to improve the performance of it.
- focus on specific promised solution that aim to provide high-performance and performance portability for python applications, specially Numba.

9.2 Python accelerators

A few popular solutions that enhance python's performance

- Numpy

- SciPy, extends the functionality of NumPy
- PyPy, a just-in-time compiler and interpreter for Python. It aims to provide faster efficient and compatible alternative implementation of Python language.
- Cython, enabling decarations of static typing to functions, variables, and classes, allows C code to be generated once and then compiles with C/C++ compilers to produce efficient C code.
- Numexpr, a module which accelerate evaluations of a numerical expression operation on NumPy.

9.3 Numba in a nutshell

9.4 Test case: matrix-matrix multiplication

10 Actor model of Anemone functional language [1]

10.1 Abstract

not be able to download yet

11 A process calculus for parallel and distributed programming in Haskell [4]

11.1 Abstract

- parallel programming and distributed programming is hard to implement, result in non-deterministic program behaviour.
- gap between model and implementation
- propose a fully determinisitc process calculus for parallel and distributed programming and implement it as a domain-specific language in Haskell to address these problems.
- achieve correctness guarantees regarding process composition at compile time through Haskell's type system.
- Their result could be used as a high-level tool to implement parallel and distributed programs.

(to read)

12 Function portability of molecular dynamics on heterogeneous parallel architectures with OpenCL [7]

12.1 Abstract

- evaluate latency, data transfer, memory access characterisitcs of parallel compute intense work
- data layout, for which the access of structure-of-arrays shows best performance in most cases.
- performance portability is a problem since various architectures strongly depend on specific vectorization optimization.

12.2 Conclusion

- One of the main goals of the present work was to investigate the performance character- istics of a function portable cell-based MD program on a set of different architectures.
- OpenCL has been chosen because it allows for interoperability on different types of architectures. Without any changes of the code it was possible to execute a benchmark on several multi- and many-core systems. Compared severl features:
    1. memory bandwidth
    2. core speed
    3. effects of data structure layout (an important issue for GPU architectures)
        - arrays-of-structures
        - structure-of-arrays
- Current their research focus on function portability, A desirable feature would be performance portability

13 Bibliography

References

[1] P. Batko and M. Kuta. Actor model of anemone functional language. *The Journal of Supercomputing*, 74(4):1485–1496, 2018.

[2] G. Bernabé, R. Hernández, and M. E. Acacio. Parallel implementations of the 3d fast wavelet transform on a raspberry pi 2 cluster. *The Journal of Supercomputing*, 74(4):1765–1778, 2016.

[3] Z. W. Bhatti, R. Wuyts, P. Costanza, D. Preuveneers, and Y. Berbers. Efficient synchronization for stencil computations using dynamic task graphs. *Procedia Computer Science*, 18(nil):2428–2431, 2013.

[4] C. Blöcker and U. Hoffmann. Pardis: a process calculus for parallel and distributed programming in haskell. *The Journal of Supercomputing*, 74(4):1473–1484, 2018.

[5] A. Braeken, P. Kumar, M. Liyanage, and T. T. K. Hue. An efficient anonymous authentication protocol in multiple server communication networks (eaam). *The Journal of Supercomputing*, 74(4):1695–1714, 2017.

[6] J. Bylina. Parallelization of stochastic bounds for markov chains on multicore and manycore platforms. *The Journal of Supercomputing*, 74(4):1497–1509, 2018.

[7] R. Halver, W. Homberg, and G. Sutmann. Function portability of molecular dynamics on heterogeneous parallel architectures with opencl. *The Journal of Supercomputing*, 74(4):1522–1533, 2018.

[8] E. M. Khaneghah, N. Shadnoush, and A. H. Ghobakhlou. A mathematical model to calculate real cost/performance in software distributed shared memory on computing environments. *The Journal of Supercomputing*, 74(4):1715–1764, 2017.

[9] A. Marowka. Python accelerators for high-performance computing. *The Journal of Supercomputing*, 74(4):1449–1460, 2017.

[10] J. Posner and C. Fohry. Hybrid work stealing of locality-flexible and cancelable tasks for the apgas library. *The Journal of Supercomputing*, 74(4):1435–1448, 2018.

[11] L. Szustak. Strategy for data-flow synchronizations in stencil parallel computations on multi-/manycore systems. *The Journal of Supercomputing*, 74(4):1534–1546, 2018.

[12] R. Thackston and R. C. Fortenberry. The performance of low-cost commercial cloud computing as an alternative in computational chemistry. *Journal of Computational Chemistry*, 36(12):926–933, 2015.

[13] P. Thoman, K. Dichev, T. Heller, R. Iakymchuk, X. Aguilar, K. Hasanov, P. Gschwandtner, P. Lemarinier, S. Markidis, H. Jordan, T. Fahringer, K. Katrinis, E. Laure, and D. S. Nikolopoulos. A taxonomy of task-based parallel programming technologies for high-performance computing. *The Journal of Supercomputing*, 74(4):1422–1434, 2018.