Parallel Performance Problems on Shared-Memory Multicore Systems: Taxonomy and Observation: Background Sources


Roman Atachiants, Gavin Doherty and David Gregg
Trinity College Dublin

In this document, we provide details on some of the sources used in deriving the initial version of the taxonomy. Please see the full paper for details of the final taxonomy. Some of the performance problems were well-known to us as programmers and teachers of parallel programming. Others come specifically from the industrial and research literature.

Task granularity
The problems around task granularity arose primarily out of our own experience of programming with OpenMP, which allows directives for parallelism to be added to C/Fortran programs. OpenMP allows the granularity of parallelism in FOR loops to be specified, and texts on OpenMP discuss some of the performance issues around granularity of FOR loops:

@book{Chandra2001,
  author = {Chandra, Robit and Dagum, Leonardo and Kohr, Dave and Maydan, Dror and McDonald, Jeff and Menon, Ramesh},
  title = {Parallel Programming in OpenMP},
  year = {2001},
  isbn = {1-55860-671-8, 9781558606715},
  publisher = {Morgan Kaufmann Publishers Inc.},
  address = {San Francisco, CA, USA},
}

Lock contention
With respect to synchronization, problems with lock contention are well known from standard parallel programming texts such as:
@book{Ben-Ari:1982:PCP:578092,
  author = {Ben-Ari, M.},
  title = {Principles of Concurrent Programming},
  year = {1982},
  isbn = {0137010788},
  publisher = {Prentice Hall Professional Technical Reference},
}

Low work-to-synchronization ratio
We learned about problems with low work to synchronization ratio from our own programming, but it is also well-known in the research literature on the design of locking algorithms, and in particular with respect to biased locking, which attempts to solve the problem for the case where most locking is done by a single thread.

@inproceedings{Russell:2006:ESA:1167473.1167496,
  author = {Russell, Kenneth and Deulefs, David},
  title = {Eliminating Synchronization-related Atomic Operations with Biased Locking and Bulk Rebiasing},
}
Lock convoy
We have never seen a lock convoy problem in our own experience of multicore software development, but they are occasionally mentioned on blogs, and in particular with reference to some older versions of Microsoft Windows.


Microsoft provides a brief online guide to performance problems that can arise in parallel applications with multiple threads. It specifically lists lock contention, uneven workload distribution, over-subscription, inefficient input output, and lock convoys as important problems.

“Common Patterns for Poorly-Behaved Multithreaded Applications”

Badly-behaved Spinlocks
Spinlocks are among the earliest developed locking mechanisms. We became aware of the performance problems that can arise with spinlocks through the articles on the website of the company Lockless Inc.

http://locklessinc.com/articles/

These performance problems are well-known to researchers who design “lock free” algorithms, but are less-known among practitioners. A good textbook on the subject is:

@book{Herlihy:2008:AMP:1734069,
  author = {Herlihy, Maurice and Shavit, Nir},
  title = {The Art of Multiprocessor Programming},
  year = {2008},
  isbn = {0123705916, 9780123705914},
  publisher = {Morgan Kaufmann Publishers Inc.},
  address = {San Francisco, CA, USA},
}

Data sharing
Performance problems around data sharing should be well understood by anyone who understands parallel computer architecture and cache coherency protocols. However, in our own personal experience of teaching parallel programming, even students who have taken a computer architecture module that deals with cache coherency in the previous semester seldom understand the performance implications for parallel software. Cache
coherency protocols are described in standard computer architecture texts (such as "Computer Architecture: A Quantitative Approach" by Hennessy and Patterson). Intel describe the practical implications for software in their excellent "Intel 64 and IA-32 Architectures Optimization Reference Manual". (see below under NUMA)

**Load Balancing**

Load balancing problems are as old as parallel computing, and in particular Gene Amdahl defined his famous "Amdahl's law" in 1967. Load balancing is a core topic of most books on parallel software development.


**Data Locality**

The idea of data locality is known to everyone with a background in computing, and caching is dealt with extensively in the industrial and research literature.

**TLB Locality**

TLB locality is less widely understood, but is well-known to computer architects and in the traditional "high-performance computing" community. TLB locality is well-documented as a performance problem for parallel matrix multiplication algorithms, where elements of successive within a column of a matrix may be distant in memory.

For example:
DRAM Paging
We originally learned about performance problems that can arise with DRAM memory pages from engineers in the embedded development section in Intel Ireland. The problem is documented in the article:
• Ulrich Drepper, What every programmer should know about memory, Publisher: Red Hat; 21. November 2007.

A good introduction to practical issues in multicore programming can be found in the following book from Intel Press. We first became aware of the performance problems that arise with false sharing of data in multicore machines from this book.

@book{Akhter2006,
  title={Multi-core programming: : Increasing Performance through Software Multi-threading},
  author={Akhter, Shameen and Roberts, Jason},
  volume={33},
  year={2006},
  publisher={Intel press Hillsboro}
}

True and False Data Sharing
Although performance problems arising from both true and false data sharing are recognized by experts in parallel software development, we had great difficulty finding a way to identify these performance problems from existing texts and research papers. We eventually found an excellent paper from researchers working on the design of multicore computer architectures.

@article{Venkataramani:2011,
  author = {Venkataramani, Guru and Hughes, Christopher J. and Kumar, Sanjeev and Prvulovic, Milos},
  title = {DeFT: Design Space Exploration for On-the-fly Detection of Coherence Misses},
  journal = {ACM Trans. Archit. Code Optim.},
  issue_date = {July 2011},
  volume = {8},
  number = {2},
  month = jun,
  year = {2011},
  issn = {1544-3566},
  pages = {8:1–8:27},
  articleno = {8},
  numpages = {27},
  url = {http://doi.acm.org/10.1145/1970386.1970389},
  doi = {10.1145/1970386.1970389},
  acmid = {1970389},
  publisher = {ACM},
  address = {New York, NY, USA},}
Competition for shared resources
The operating systems community are aware of problems arising for competition between threads for shared resources, but we have not seen the problems well documented for general parallel software development. We originally learned about the problem when trying to improve performance in a parallel program using better scheduling techniques, and we read the following:

@inproceedings{Zhuravlev:2010:ASR:1736020.1736036,
    author = {Zhuravlev, Sergey and Blagodurov, Sergey and Fedorova, Alexandra},
    title = {Addressing Shared Resource Contention in Multicore Processors via Scheduling},
    booktitle = {Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems},
    series = {ASPLOS XV},
    year = {2010},
    isbn = {978-1-60558-839-1},
    location = {Pittsburgh, Pennsylvania, USA},
    pages = {129--142},
    numpages = {14},
    url = {http://doi.acm.org/10.1145/1736020.1736036},
    doi = {10.1145/1736020.1736036},
    acmid = {1736036},
    publisher = {ACM},
    address = {New York, NY, USA},
    keywords = {multicore processors, scheduling, shared resource contention},
}

General References:
Although our research assumes a generic multi-threading programming model, it is also useful to consider common patterns of parallelism in real programs. Intel provide an excellent guide to patterns of parallelism that suit their “Threaded Building Blocks” library for multicore computation, which highlights some of the performance issues that can arise.

@techreport{Intel2010,
    institution="Intel Corporation",
    title="Intel Threading Building Blocks Design Patterns, Document Number 323512-003US",
    month = "September",
    year=2010}

Intel produces an excellent software performance manual: “Intel 64 and IA-32 Architectures Optimization Reference Manual”, which has a chapter dealing with multicore performance. This manual deals with many issues around locks, spinlocks, data sharing and differences in access times for non-uniform memory access time (NUMA) parallel computers.

@book{inteloptimize,
    author = {{Intel Corporation}},
}
Performance analysis tools for HPC

There has been a great deal of prior research on performance analysis tools for traditional high-performance computing. Much of this work is aimed at large supercomputers with hundreds or thousands of processors and distributed memory. The typical programming models are message passing (using MPI) and annotation for parallel for loops (OpenMP, at least prior to version 3.0). Two useful references for these tools are:

@article{Adhianto2010,
author = {Adhianto, L and Banerjee, S and Fagan, M and Krentel, M and Marin, G and Tallent, N R},
journal = {Context},
keywords = {binary analysis, call path profiling, execution monitoring, performance tools, tracing},
pages = {1--7},
title = {{HPCToolkit: Tools for performance analysis of optimized parallel programs}},
year = {2010}
}

@inproceedings{Burtscher2010,
author = {Burtscher, Martin and Kim, Byoung-Do and Diamond, Jeff and McCulpin, John and Koesterke, Lars and Browne, James},
title = {{PerfExpert: An Easy-to-Use Performance Diagnosis Tool for (HPC) Applications}},
booktitle = {Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis},
series = {SC '10},
year = {2010},
isbn = {978-1-4244-7559-9},
pages = {1--11},
numpages = {11},
url = {http://dx.doi.org/10.1109/SC.2010.41},
doi = {10.1109/SC.2010.41},
acmid = {1884700},
publisher = {IEEE Computer Society},
address = {Washington, DC, USA},
}
Right now I am learning about parallel programming in C with openmp and now I have stumbled upon the following problem. I have a simple for loop which I want to parallelize. Using openmp, I thought the following code should do the job. unsigned long int teller_groot; int boel = 0; int k = 0; int l = 1; unsigned long int sum2; int thread_id; int nloops; #pragma omp parallel private(thread_id, nloops) {. sum2 = 0; #pragma omp for. for (teller_groot=1000000; teller_groot<1000000000000; teller_groot++) {.