Have you used Multi-Core for speed? What did you do and was it worth the effort?

Asked 15 years, 9 months ago Modified 8 years, 1 month ago

Viewed 848 times

1

So I did not look at the right location before posting this..

```
I was looking at the result of the computer
language benchmark game:

<http://shootout.alioth.debian.org/u32q/index.php>

And it seems that most of the fastest solutions
are still C/C++ using just
a single core of the 4 core machine that runs the
tests.

I was wondering if multi-core is worth it at all
for single tasks or if you
really need some more speed just tune up your
code, rewrite in C/C++ instead.
```

When you click on the full benchmark link like: http://shootout.alioth.debian.org/u32q/benchmark.php?test=knucleotide&lang=all it is obvious that quite a few solutions use multiple core.

It would still be interesting to hear of your personal experiences:

Have you had success using 4 or 8 cores in order to actually improve performance on a single task?

What tools/language did you use?

How big was the improvement?

Was it worth the effort?

performance multicore

Share

Improve this question

Follow

edited Mar 10, 2009 at 19:31

community wiki
5 revs, 2 users 100%
Jeroen Dirks

11 Answers

Sorted by: Highest score (default)

3

> And it seems that the fastest solutions are still C/C++ using just a single core of the 4 core machine that runs the tests.

No, that's not true for all codes. In fact, of the codes I've looked at, all use multiple parallel threads, and thus multiple cores. In fact, some (e.g. k-nucleotide) use fancy

architecture like OpenMP (or, also interesting, SSE parallelization) to help parallelization.

EDIT In fact, the fastest C++ solution for **every** problem uses parallel threads, with three exceptions:

1. fasta benchmark, hard (but altogether possible) to parallelize due to random generator usage.
2. pidigits, uses the GMP library.
3. n-body, could be parallelized.

… and most other solutions also use SSE2 support.

Share Improve this answer Follow

edited Mar 11, 2009 at 18:13

answered Mar 10, 2009 at 18:58

Konrad Rudolph
545k ● 139 ● 956 ● 1.2k

Neat, I missed that k-nucleotide solution. – Jeroen Dirks Mar 10, 2009 at 19:08

See my update. In fact, almost all solutions use parallel threads (either as pthreads or via OpenMP).
– Konrad Rudolph Mar 10, 2009 at 19:21

igouy: n-body at least uses SSE parallelization. But true, this could also be parallelized across all four cores rather easily.
– Konrad Rudolph Mar 11, 2009 at 10:18

the OP asked about multi-core not SSE. "parallelized across all four cores rather easily" - it's always easier to say rather

than do :-) meteor-contest also (but that's a contest and not usually included) – igouy Mar 11, 2009 at 14:59

igouy: yes, I wasn't questioning the truth of your correction. I should have amended my answer. – Konrad Rudolph Mar 11, 2009 at 18:12

2

In order to increase performance for a single task on a multicore system, you'd have to design your task to split up into different parts (ala mapreduce) and hand each part to a different core. Lots of programs do things like that, and it does increase performance.

A few compression algorithms currently support more than one processor, such as 7zip. It's not terribly difficult to do something like that, but if your task can't break into cooperative parts, you're not going to get any help from more than one core.

Share Improve this answer

Follow

answered Mar 10, 2009 at 18:54

Alex Fort

18.8k ● 5 ● 44 ● 51

Notice that the Google MapReduce infrastructure (or even the more general map/reduce schema) is just one of many possibilities. – Konrad Rudolph Mar 10, 2009 at 18:59

7zip is limited to how much it can do with multiple cores (there was a discussion about this on Jeff's blog: codinghorror.com/blog/archives/001231.html) Apparently, 7zip doesn't split a file into blocks... – Powerlord Mar 10, 2009 at 19:24

1

It really depends on how the algorithm works and the size of the dataset you're processing as to whether it scales well across multiple cores. Staying on the same core gives you an awful lot of advantages, including taking advantage of processor pipelining and using registers and cache - all of which are super-quick.

As multiple core become more important in the future, we'll probably see some interesting cross-core optimizations becoming available.

Share Improve this answer

Follow

answered Mar 10, 2009 at 18:59

James L

16.8k ● 10 ● 55 ● 72

0

How do you define a "single task"? Many single conceptual tasks can nevertheless be split up into many independent subtasks. That's where multiple cores *may* provide a performance boost.

Of course, this requires you to actually structure your program so that these subtasks are actually able to be processed independently.

Share Improve this answer

Follow

answered Mar 10, 2009 at 19:05

Stack Overflow is garbage

247k ● 52 ● 353 ● 555

0

I use multicore quite regularly when performing monte carlo simulations. In this case it can be an absolute godsend, because sometimes these simulations take forever and each run is independent of every other run. In fact, right now I'm waiting for a monte carlo simulation to run on my quad core.

Another use case is when testing a machine learning algorithm using cross-validation. The dataset can be loaded once and stored in an immutable object. Then, each cross-validation iteration can be performed independently. For things like this, the key is to be careful about memory allocations and avoid the implicit lock acquisition this involves. If you allocate and free/garbage collect infrequently enough, the speedup can be near linear in cores used.

Share Improve this answer Follow

answered Mar 11, 2009 at 2:59

community wiki
dsimcha

0

I've seen 4x improvement easily gained on a processing pipeline.

Share Improve this answer Follow

answered Mar 11, 2009 at 3:21

community wiki
Joshua

0

There are certain types of tasks which can be trivially multithreaded, and thus, allow a performance increase in systems which have multiple cores.

Image processing is one arena which could benefit from multicore. For example, applying an image filter is a process that is independent of results from other parts of an image. Therefore, as mentioned previously in Alex Fort's answer, by splitting the problem, in this case, the image filtering, into multiple parts and running the processing in multiple threads, I was able to see a decrease in processing time.

In fact, multithreading increased the performance not only on multicore processors, but also on my Intel Atom N270-based system, which only has a single core but offers two logical cores through simultaneous multithreading (hyper-threading).

I performed a few tests of applying an image filter using multiple threads (by splitting the processing into four threads) and a single thread.

For multithreading, the `ExecutorService` from the `java.concurrent` package was used to coordinate the multithreaded processing. Implementating this functionality was fairly trivial.

Although not exact numbers, nor an perfect benchmark, on a dual-core Core 2 Duo, the processing time for multithreaded code decreased by 30-50% compared to single-threaded code, and on a hyper-threaded Atom, a decrease of 20-30%.

As with other problems which involve splitting the problem into parts, the scalability of this method of processing is going to depend on the time spent by the steps where the problem is split up and combined.

Share Improve this answer

Follow

edited May 23, 2017 at 12:30

Community Bot

1 ● 1

answered Mar 11, 2009 at 3:46

coobird

161k ● 35 ● 214 ● 227

0

I wrote a CD label editing program in REALbasic which was cross-platform (hence not being able to just rely on GDI+ or Cocoa). It allows layering of multiple masked images with clipping to label shapes.

I switched from the image blit and zooming routines built into the language to using a plugin which was able to use up to 4 cores and achieved a significant speedup of key user operations, especially when zoomed.

This was a nice domain separation for a drop-in solution - I passed in a single image to the binary plugin and it

internally partitioned the work across the processors. As a library solution, it required no multi-core awareness on the part of my program.

Share Improve this answer Follow

answered Mar 11, 2009 at 4:30

community wiki
Andy Dent

0

make -j 6

Took 6 minutes out of a 7 minute build. :)

Share Improve this answer Follow

answered Mar 11, 2009 at 4:34

community wiki
dicroce

0

I would like to remind everyone of Amdahl's Law, which is a description of the diminishing returns gained by increased parallelism, and serves also to model how much speedup can be expected for a given algorithm.

Share Improve this answer Follow

answered Mar 11, 2009 at 4:37

community wiki

Paul Nathan

0

I have reached a high speedup using 16 cores (in an Amazon EC2 instance) in a project to solve SVMs, my speedup goes from 10x to 16x depending on the dataset that the algorithm uses:

https://github.com/RobeDM/LIBIRWLS

This is the paper I wrote:

http://www.sciencedirect.com/science/article/pii/S0167865516302173

Share Improve this answer

Follow

answered Nov 3, 2016 at 21:11

community wiki
Rob