Open Parallel

Software for Multicore and Parallel Computing

TBB in Wordpress

White Paper – October 2010

Introduction

When we at Open Parallel started to look at a way to implement **TBB** into **PHP** and **HipHop**, we initially planned to tackle the HipHop optimizer and try to come up with some sort of automatic conversion that uses TBB where possible. This turned out to be a not workable task for the moment and we went down the track of **mapping TBB commands directly to new PHP functions.**

The idea was to implement TBB commands into HipHop and write PHP equivalents for the case where either HipHop is not compiled with our patches or where the project runs on a more traditional stack.

First command: parallel_for

We started out with parallel_for as it looked like one of the commands that suit a web project best. There are many loops in WordPress and the ability to effect free parallelization on them looks like a very promising option. As not all loops can be run in parallel, we opted for writing a patch set for WordPress that used the new syntax and commands.

On the HipHop side, TBB was already used for memory management and adding the parallel_for command was pretty straightforward. We extended the known language constructs of HipHop by the command parallel_for and created a patch set for the HipHop code base. With these patches in place, we were able to compile PHP constructs containing the keyword parallel_for on the HipHop engine.

After initial tests with the new command and demo scripts on how to use it, we took the WordPress code apart and tried to find a foreach loop that was used frequently. We found that the plugin code had one of the more frequently used foreach loops and rewrote it using our new parallel_for – it worked after some debugging.

Verdict so far: extending HipHop is not entirely straightforward but can be done.

Adding TBB to the PHP language is possible, but the developer using it needs to judge if a loop can run in parallel or not. The language construct we currently use might not be straightforward enough for the junior PHP developer, but then again, they will most probably not be our target audience anyway, as they have to learn how to judge if something can run in parallel first.

Bottom line: An intermediate PHP developer can use the TBB extension after some initial training on what to look for.

An example invocation of the parallel_for function looks like this:

```
function test_pfor_callback($indices) {
foreach ($indices as $idx) {
echo “{$idx}\n”;
flush();
}
}
parallel_for(0, 1000, ‘test_pfor_callback’);
```

WordPress on HipHop

The next step was to run the hacked WordPress version on HipHop and compare it to the original version. This bit was the impressive bit. Just to recap, we added one command out of the TBB functionality to PHP and then modified one foreach to run with our parallel_for implementation.

In the initial tests, we saw a significant drop in memory usage and a sizable reduction of the time our test suite needed to run. We subsequently moved more code into parallel_for constructs and refined our testing, and even though our gains in terms of memory dropped, we managed to gain more on the execution time.

Adding TBB

The lessons learned from this experiment are interesting. HipHop is a better way to run PHP – we kind of expected this – but adding TBB to the game for more than the memory management looks like the way forward. **TBBs parallel_for gave us a considerable speed-up and a significant decrease in memory consumption**

This experiment also showed that there is a vast field of optimization in script languages for optimizing loops. We will look into this in more depth and try to come up with a **more general TBB implementation in PHP and HipHop in the future**. The reduced memory footprint and the decrease of execution time due to running tasks in parallel are addressing very real life problems, and fixing this in a more general way looks like a very much needed language extension.

Details of the Tests

Going into the details of the tests, we explain our test set-up and show how these tests can be replicated on your hardware to see what speedups you can gain – the first tests ran on real networks on different physical machines whereas the later ones ran with the one described below.

We tested on virtual machines running VirtualBox with HipHop and WordPress (including MySQL) running in the VM and Tsung running on the host machine. The host was running MacOS 10.6.4 and the nodes were running Ubuntu 10.04. For each test the VM was rebooted, HipHop started with our Open Parallel startup script and then Tsung started from the host.

Collectd was used to gather the performance data from the node systems and no other services as the required ones were running at the time of the tests, in order to reduce noise. We even stopped using Nginx as a frontend loadbalancer to make sure we were really only measuring the pure time HipHop needs to render the page.

The Open Parallel WordPress repository can be found on GitHub and there are some repositories that hold additional resources that might be needed to run the tests or come in handy for testing. We also published our Tsung config to illustrate how our typical user session looked like that we ran our tests with.

Memory Usage Reduction

Testing revealed that we were able to reduce the memory usage by about 100MB in our current version. This value moves quite a bit depending on the functions that are moved into the parallel_for construct and it is quite hard to come up with a MB/invocation number. We did some in-depth testing in that area and all we can say is that the amount of memory used was **always below** the amount the unpatched version used – from time to time not by much – but always below.

The parameter that was constantly below the unpatched version – and by a lot – was the connection time and the number of sessions handled in parallel. We were able to optimize in that area to the extend of more than one minute less time needed to complete our test run, and managed to get the amounts of pages served per second up to 7.2 compared to 6.3 in the unpatched version.

The attached graphs show the memory and CPU usage we recorded during the test runs and show the responses/sec for the different setups.

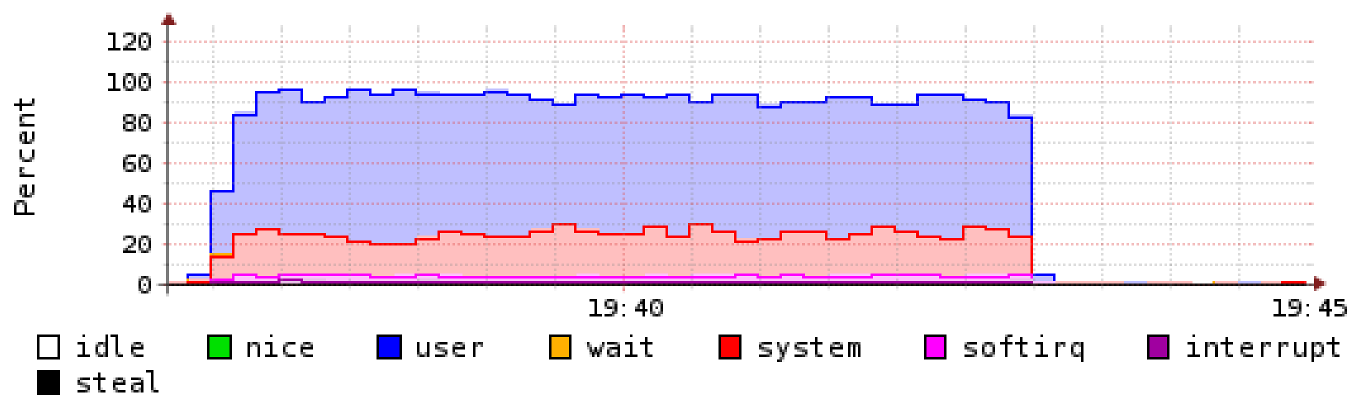
We are happy to supply more information if needed to replicate the tests on your systems and are happy to discuss the outcomes on the **HipHop mailing list.**

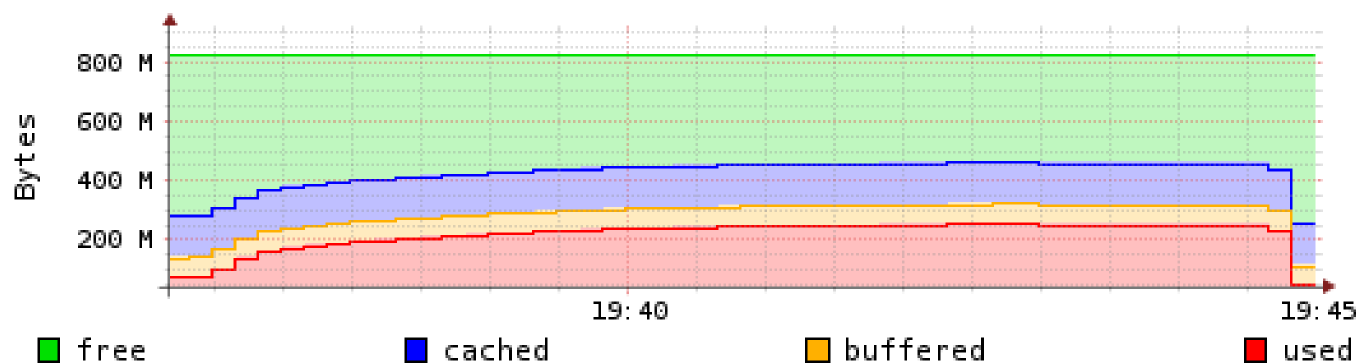
© **Open Parallel** White Paper
November 2010

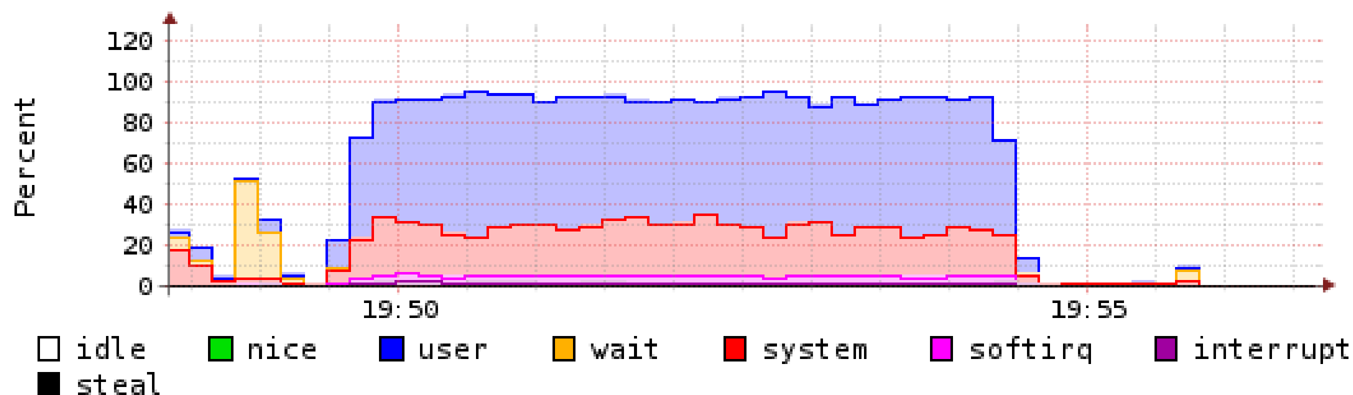
www.OpenParallel.com

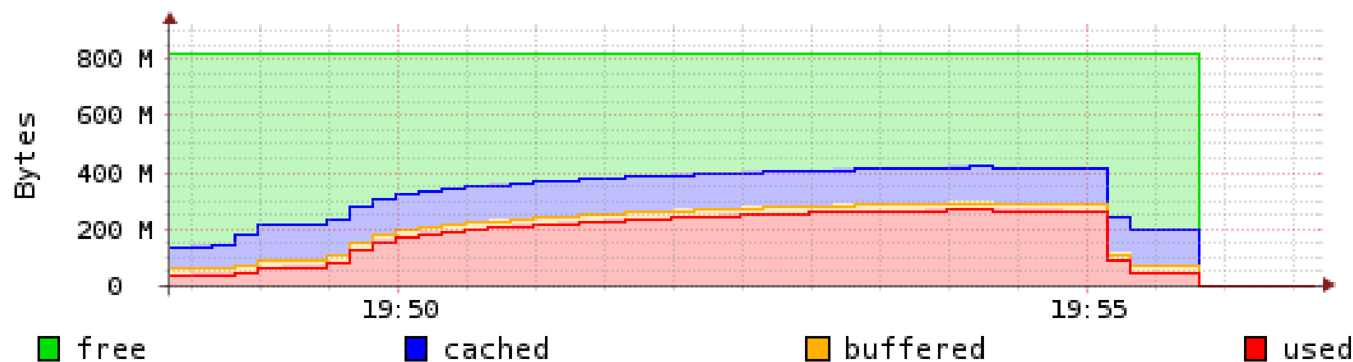
Correspondence: Nicolas.Erdody@OpenParallel.com

CPU and Memory Graphs:

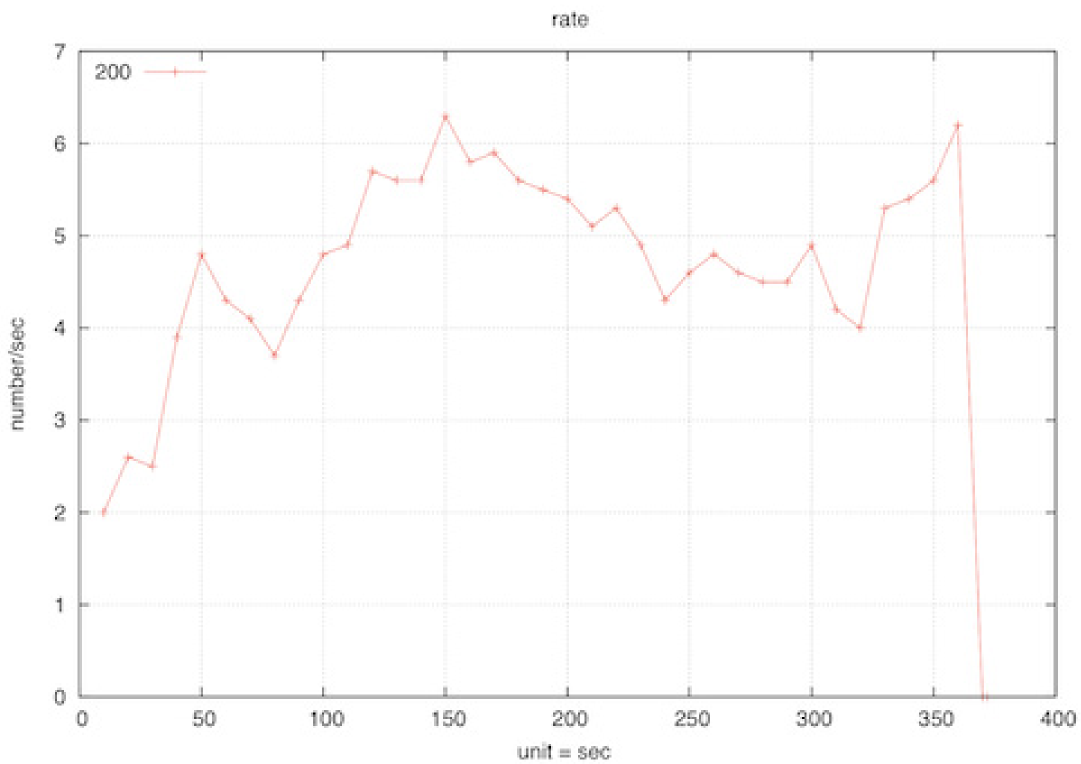




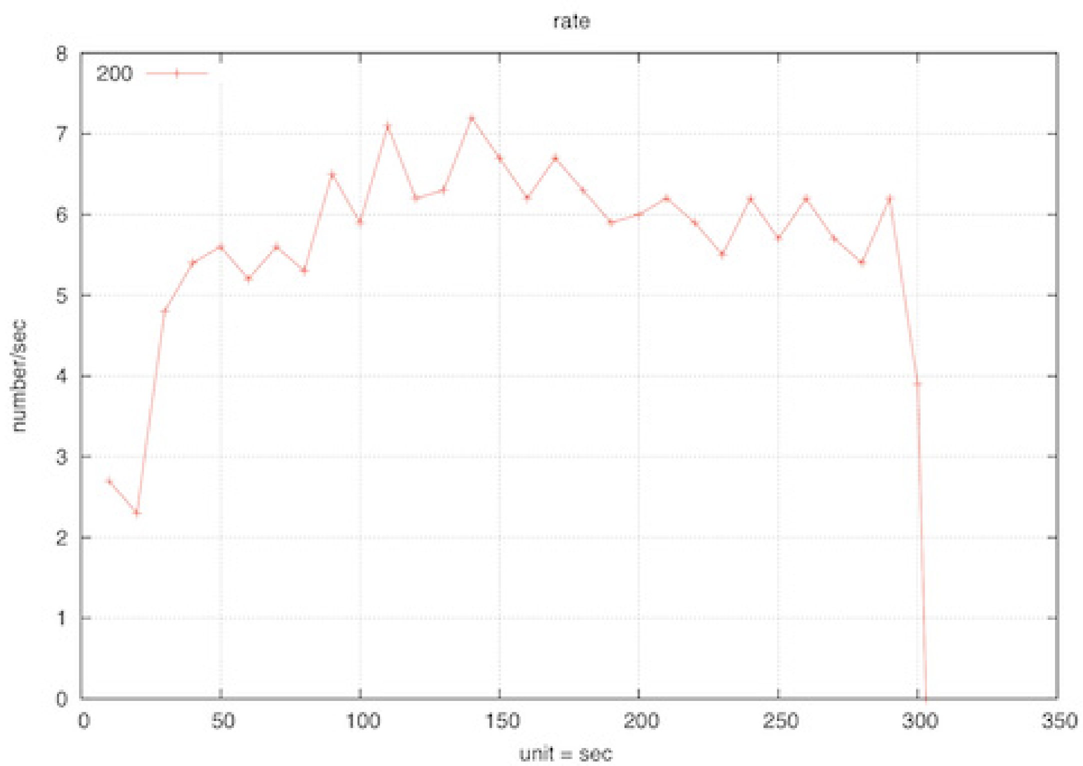




Pages / seconds Graphs:



Pages - Seconds - Standard



Pages - Seconds - Optimized

© **Open Parallel** White Paper
November 2010

Correspondence: Nicolas.Erdody@OpenParallel.com