# D5.4– Report on Tuned Linux-ARM kernel and delivery of kernel patches to the Linux kernel Version 1.0

## Document Information

| | |
|---|---|
| Contract Number | 288777 |
| Project Website | www.montblanc-project.eu |
| Contractual Deadline | M18 |
| Dissemintation Level | PU |
| Nature | Other |
| Coordinator | Alex Ramirez (BSC) |
| Contributors | Roxana Rusitoru (ARM) |
| Reviewers | Isaac Gelado (BSC) |
| Keywords | Linux kernel, HPC, Transparent HugePage |

***Notices:*** *The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 288777*

## Change Log

| Version | Description of Change |
|---|---|
| v0.1 | Initial draft released to the WP5 contributors |
| v0.2 | Revised draft released to the WP5 contributors |
| v1.0 | Version to send to the WP leaders and the EC |

# Contents

# Contents

# 1 Introduction

We aim to create an optimized software stack tailored to an ARM-based HPC system. As a result, we are looking at exploiting OS features that can improve performance.

We investigate the effects of using hugepages through Transparent HugePages on a number of benchmarks and sample HPC applications, whilst running on the MontBlanc chosen SoC: Exynos 5 Dual. We are presenting results for both pandaboard, and the Arndale.

# 2 Analysis of Transparent HugePages: pandaboard

## 2.1 Introduction

The aim of this task is to investigate the effects of using hugepages on a number of benchmarks and sample HPC applications, whilst running on existing SoCs. For this purpose, we are using an internal Linaro Linux kernel 3.6 using an userspace from Linaro Ubuntu 12.08 (originally running with a kernel version 3.4). The advantage of this kernel is that it has support for HugeTLB and THP.

At this stage, we are only investigating Transparent HugePages, as to offer preliminary results on the usage of hugepages. A more comprehensive analysis, including HugeTLB, will happen on the Arndale board as results will be closer to what we are expecting in the final system.

## 2.2 Hardware setup

We ran all experiments on a pandaboard [1] Rev A3 (1GHz, 32 KB L1 cache with a separate instruction and data cache, and 1 MB L2 cache). The system has 1 GB LPDDR2 RAM. Later, we added another pandaboard (Rev A1) using an identical setup to the first one. Both have their rootfs served over NFS from the same server, including the `/home` directories. The connection to the server is made via 100 Mbps Ethernet, through a switch shared with a number of other devices.

### 2.2.1 Limitations

The main limitation with the pandaboard is the fact that its PMU does not have dedicated IRQs.

## 2.3 Benchmarks

In order to get a comprehensive image of the kernel's impact and of how we perform compared with other HPC systems, we used a wide variety of benchmarks that test from the kernel noise to memory bandwidth. We used the following benchmarks [2, 3, 4]:

- **FTQ/FWQ** (Fixed Time Quanta/Fixed Work Quanta) - these are benchmarks for measuring the OS noise. They calculate how many simple operations (increment) you can perform in a particular time quanta, or how long it takes to perform a particular work quanta.
- **STRIDE** - it is a suite of benchmarks designed to severely stress the memory subsystem. They utilize combinations of loops of scalar and vector operations and measure the MFLOP rate delivered as a function of the memory access patterns and length of vector utilized. All benchmarks use double precision variables.
  - Strid3 - this benchmark accesses an array wih varying stride, up to strides of 1024. The stride also guides the length of the array that is accessed, of $256 * stride$. The core operation that occurs in the main loop is `y[i] += t * x[i]`. Strid3 has a C and Fortran version.
  - Vecop - contains a number of vector operations, as follows:
    - V1S1M3 - `y[i] += t1 * x[i]`
    - V1S2M3 - `y[i] = t1 * y[i] + t2 * x[i]`
    - V1S3M3 - `y[i] = t1 * (t2 * y[i] + t3 * x[i])`
    - V2S2M3 - `y[i] += t1 * (y[i] + t3 * x[i])`
    - V2S2M4 - `y[i] += t1 * (z[i] + t2 * x[i])`
    - V1S1I3 - `y[idx[i]] += t1 * x[idx[i]]`

    All arrays are accessed with a stride of 1. The variation comes in the length accessed, which starts at 4 and goes up to 1024, in increments of 4. Vecop has a C and Fortran version.
  - Cache - this benchmark varies the loop length, starting from 8 in increments of `N` up to `N * M`, where `N = 64 and M = 4096`. The core operation is `y[i] += t * x[i]`.Cache has a C and Fortran version.
  - STRIDDOT - this benchmark is similar to Strid3, as it accesses an array with increasing stride `INCX`, and the array length depends on the stride proportionally: `N * INCX`. `INCX` takes values from 1 to `M`. For this benchmark `N = 256 and M = 1024`. The core operation is multiplication of two vectors and increment of scalar: `DOT = DOT + X(I) *Y(I)`. STRIDDOT has only a Fortran version.
  - CACHEDOT - this benchmark is similar in nature with Cache, that it only varies the loop length in increments of 8, starting from `N`, up to `N * M`, where `N = 16 and M = 1024`. The main difference is the operation, which involves

the multiplication of two vector elements and increment of a scalar: `DOT = DOT + X(I) * Y(I)`. The arrays are accessed with stride 1. CACHEDOT has only a Fortran version.

- **GUPS** (giga-updates per second) - it offers a measurement of how frequently a computer can issue updates to randomly generated RAM locations. GUPS measurements stress the latency and especially the bandwidth capabilities of a machine.
  - Vanilla - standard GUPS implementation, which needs a power of two number of processes.
  - NonPow2 - standard GUPS implementation which supports both power of two and non-power of two number of processes.
  - Opt - optimized GUPS implementation, which aims to reduce cache aliasing, has loop unrolling and other code optimizations. It needs a power of two number of processes.
- **HPL** (high performance Linpack) - solves a (random) dense linear system in double precision arithmetic on distributed-memory computers.
- **ARM HPCC** - it is an internal, mpi-free version of HPCC [5]. It is a suite of 7 benchmarks targeting aspects that can affect an HPC system (from floating point rate of execution, to sustainable memory bandwidth)
  - dgemm-blas - DGEMM performs an operation of type `C = alpha * op(A) * op(B) + beta * C`, where `op(X) = X or op(X) = X'`. ARM_HPCC dgemm-blas uses an ATLAS implementation and makes the following call: `cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans, n, n, n, alpha, A, n, B, n, beta, C, n);`. `A`, `B` and `C` are randomly generated matrices, whilst `alpha` and `beta` are taken as the initially generated middle values of the `A` and, respectively `B` arrays. `n` is either derived from user input or from preset values of `N`. `CblasColMajor` and `CblasNoTrans` are ATLAS-defined constants. `A`, `C`, `C`, `alpha` and `beta` are of type `double`, whilst `n` is an `int`.
  - dscal-blas - DSCAL scales a vector by a constant. It makes the following ATLAS call: `cblas_dscal(N,s,A,1);`. Just like for DGEMM, `N` is provided as user input else preset values are used. `A` and `s` are initialized as `A[i] = i * 1.1f` and `s = 2.1`. All variables, but `N` which is an `unsigned int`, are of type `double`.
  - max-flops-dp - this test is similar to dgemm, however it changes one of the parameters it calls `cblas_dgemm` with. It makes the following call: `cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans, l_n, l_n, l_n, alpha, A, l_n, B, l_n, beta, C, l_n);`. The only parameter that differs from DGEMM is `n`, which now is `l_n`. `A`, `B`, `C`, `alpha` and `beta` are of type `double`, whilst `l_n` is of type `long`.

- max-flops-sp - this is the single precision version of max-flops-dp.
- sgemm-blas - this is the single precision version of dgemm.
- scal-blas - this is the single precision version of dscal.
- stream-c - this benchmark uses at its core the following operation: `a[i] = b[i] + scalar * c[i]`. The variables are initialized as: `a[i] = 1.0, b[i] = 3.0, c[i] = 4.0, scalar = 3.0`. The size of the array is `N`. All variables, but `N` which is an `int`, are of type `double`.

### 2.3.1 Setup

#### 2.3.1.1 Compilation flags

All benchmarks, but FTQ/FWQ were built with the following compilation flags:

```
-O3 -mcpu=cortex-a9 -mtune=cortex-a9 -mfloat-abi=hard -mfpu=neon
          -funsafe-math-optimizations -DL2SIZE=1048576
```

When building FTQ/FWQ, `-O3` would actually cause the inner loop of the benchmark to be optimized away. As a result, we used flags suggested by the benchmark for other architectures (adapted for the pandaboard):

```
-O1 -ffast-math -funroll-loops -fexpensive-optimizations -mcpu=cortex-a9
                        -mtune=cortex-a9 -static
```

#### 2.3.1.2 Processor affinity

For the purpose of these benchmarks, we have not set the processor affinity for any benchmark, therefore it defaults to whether the benchmark sets it or not.

#### 2.3.1.3 ATLAS

In order to obtain the most performance gain out of benchmarks such as HPL and ARM HPCC, we used ATLAS (Automatically Tuned Linear Algebra Software) [6], instead of BLAS (basic linear algebra library). ATLAS is an optimized BLAS library, which auto-tunes itself to get best performance for the architecture it is running on.

## 2.4 Sample HPC application - bigDFT

Benchmarks give a good indication of each optimization's impact, due to their specialized nature. However, this is not a guarantee that real HPC applications are going to obtain similar performance improvements, or whether the optimizations are even relevant. As a result, we explored the work in the context of a HPC application, bigDFT.

Preliminary results showed no difference in performance for this benchmark. More details and further analysis will only be included for the final chip.

## 2.5 Methodology

For benchmarking the system, we used the following method:

1. Boot the baseline system with no hugepage support.
2. Run the benchmarks by using the custom-made scripts, with well-documented run parameters.
3. Enable Transparent HugePage support by enabling `CONFIG_TRANSPARENT_HUGEPAGE` and `CONFIG_TRANSPARENT_HUGEPAGE_ALWAYS`.
4. Re-run the benchmarks as before, by using the same scripts and parameters.

A list of what parameters were used for each benchmark can be found below:

- FTQ/FWQ
  - Number samples: 100000
  - FTQ iterations: 20
  - FWQ work quanta: 15
  - Number of threads: 1
- STRIDE
  - Number of threads: 1
  - All other parameters were default, program built-in ones.
- GUPS
  - N - length of global table is $2^N$: 26
  - M - number of update sets per processor: 1000
  - chunk - number of updates per set: 10000
- HPL
  - number of problem sizes (N): 1
  - problem size: 10000
  - number of block sizes (NBs): 1
  - block size: 32
  - PMAP process mapping: 0
  - number of process grids: 1
  - Ps: 1
  - Qs: 2

  - threshold: 16.0
  - number of panel factorisations: 1
  - PFACT: 1 (Crout)
  - number of recursive stopping criterium: 1
  - NBMINs: 2
  - number of panels in recursion: 1
  - NDIVs: 2
  - number of recursive panel factorisations: 1
  - RFACTs: 2 (Right)
  - number of broadcasts: 1
  - BCASTs: 0 (1rg)
  - number of lookahead depths: 1
  - DEPTHs: 1
  - SWAP: 2x (mix)
  - swapping threshold: 64
  - L1 in no-transposed form (0)
  - U in no-transposed form (0)
  - Equilibration: yes (1)
  - memory alignment in double: 8

- ARM HPCC
  - dgemm-blas
    * base: 10
    * first exponent: 6
    * last exponent: 7
    * iterations: 1000
  - dscal-blas
    * base: 10
    * first exponent: 6
    * last exponent: 7
    * iterations: 1000
  - max-flops-dp
    * N: 100
    * iterations: 1000
    * runs: 100

- max-flops-sp
  - N: 100
  - iterations: 1000
  - runs: 100
- sgemm-blas
  - base: 10
  - first exponent: 6
  - last exponent: 7
  - iterations: 1000
- sscal-blas
  - base: 10
  - first exponent: 6
  - last exponent: 7
  - iterations: 1000
- stream-c
  - base: 2
  - first exponent: 15
  - last exponent: 19
  - iterations: 1000

## 2.6 Results

In this section we cover results of running the benchmarks on the baseline and on the THP-enabled kernel. We present the results as relative performance improvement of hugepages over baseline, with all graphs being normalized based on the stock case for each benchmark individually. A comparison between the two sets of results is offered in Observations 2.6.1 and in each figure's caption.

For all STRIDE results, the measurements were taken when the performance stabilized for high strides or loop lengths.

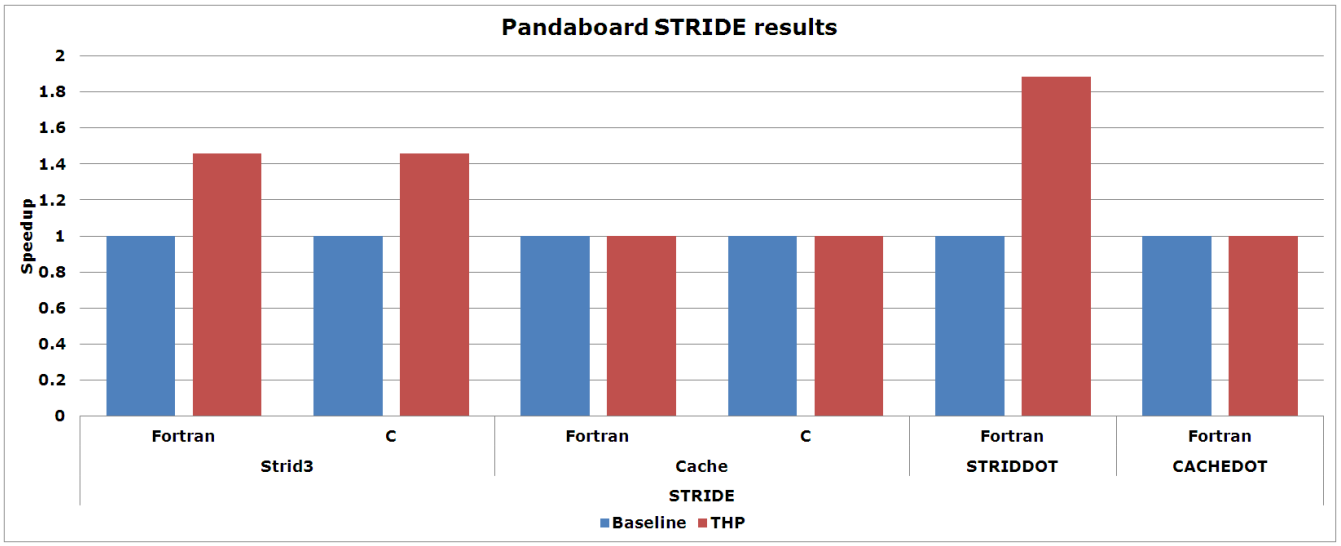


Figure 1: This graph shows us the STRIDE Strid3, Cache, STRIDDOT and CACHEDOT performance improvement of hugepages over normal ones. We observe that in all cases where we vary the stride there is an improvement varying from 1.4x up to 1.9x whilst using THP. Whenever we only vary the loop length, we observe no improvement, nor any performance degradation.

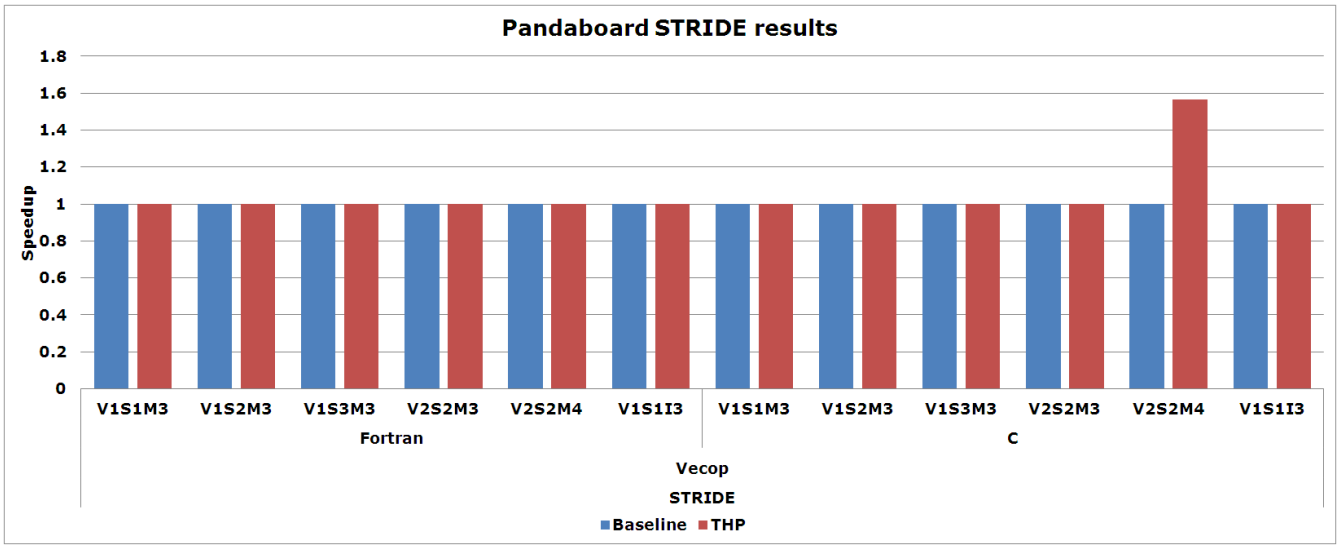


Figure 2: Here we can see the STRIDE Vecop benchmark. As the parameter that varies is the loop length, most benchmarks show no difference whatsoever. Nevertheless, V2S2M4 shows a nearly 1.6x improvement when using hugepages. This benchmark has the most memory operations, it accessing 3 arrays every iteration. In more detail, V2S2M4 has about 3 MB of array accesses, which means in the baseline case it needs 771 pages, which would thrash the TLB. However, this is not an issue in the THP case, as it would only take up 2 hugepages.

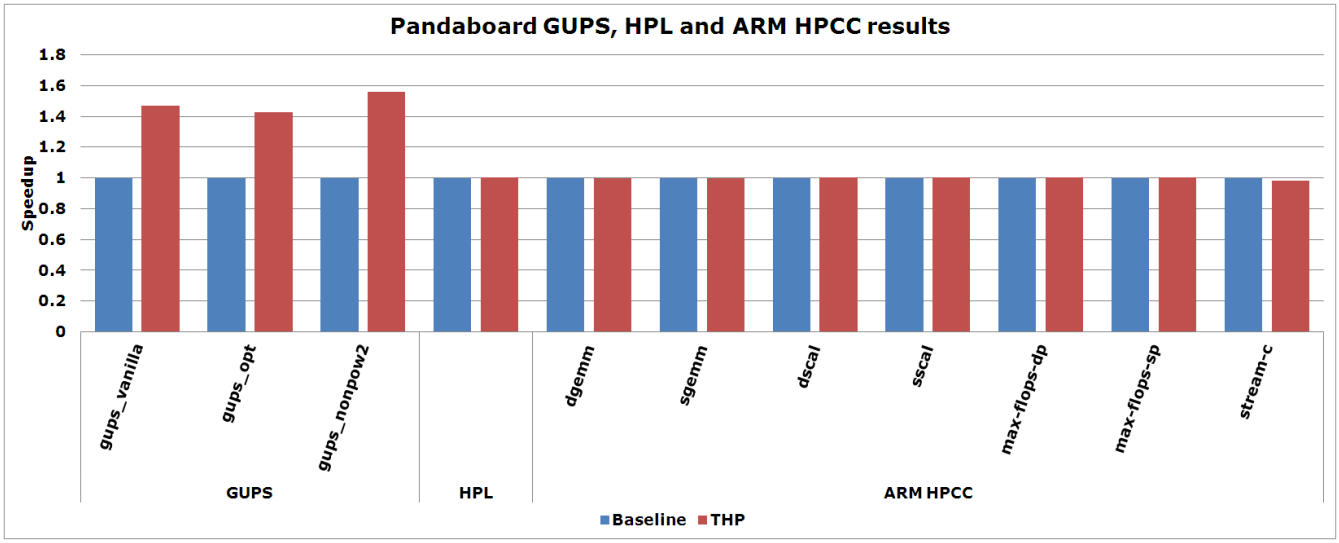


Figure 3: This figure shows us the results for GUPS, HPL and ARM HPCC. We observe that HPL shows no performance improvement when enabling hugepages. This is due to the benchmark having good data locality. Similarly, HPCC, including stream, shows no difference between the baseline and the THP cases. This is potentially due to all the benchmarks being computationally bound. However, as a benchmark which is memory bound, GUPS shows us clearly the effect of hugepages, as performance increase across the 3 variations of GUPS ranges from 42.51% to 55.91%. The smallest performance improvement is for the optimized version, whilst the largest is for the one which supports non-power of 2 number of processes.

### 2.6.1 Observations

By analysing all the results, there is a main reoccurring behaviour which is that generally benchmarks that suffer from low or no data reuse and random data accesses can now render higher performance by using hugepages. In the best case scenarios when there is data reuse or just little amounts of data, there is no performance increase. There have been no significant slow-downs observed. After looking more in-depth at the kernel noise generated by THP using FTW/FWQ, we observed little impact. Further analysis will be performed on the final SoCs.

A more detailed explanation of the results can be found in the figures' captions.

## 2.7 Conclusions

By what we have observed so far regarding the lack of significant performance decrease and the overall improvement in worst case scenarios, the low kernel noise impact, we believe THP should be used for HPC purposes.

## 2.8 Further work

### 2.8.1 Kernel profiling

In order to gain an even better understanding of the results and of what occurs in the kernel, the next step is to profile it in depth. The approach we are taking for this is to use tools such as DS-5 with Streamline, ftrace directly and trace-cmd with kernelshark. We have investigated lltng (Linux Trace Toolkit - next generation), however, due to the pandaboard's hardware limitations, the tool did not work.

### 2.8.2 In-depth analysis of results

We attempted to perform a more in-depth analysis of results, however, the pandaboard's hardware limitations have made it very difficult. Given the technical difficulties and the fact that we are using pandaboards, which are not the final SoCs, we have decided to not spend much time on this analysis at this point.

### 2.8.3 Sample HPC applications

For this main experiment we only ran bigDFT with and without hugepages and observed no change. We will run more comprehensive experiments and analyse the results in more depth on the final chip, as due to the exhaustive nature of the benchmarks, we got a conclusive result for THP on the pandaboard.

# 3 Analysis of Transparent HugePages: Arndale board

## 3.1 Introduction

The aim of this task is to investigate the effects of using hugepages on a number of benchmarks and sample HPC applications, whilst running on the MontBlanc chosen SoC: Exynos 5 Dual. For this purpose, we are using an Arndale board, running a Linaro Linux kernel 3.7 using an userspace from Linaro Ubuntu Server 13.01. The advantage of this kernel is that it has support for HugeTLB and THP, power management and perf events.

At this stage we am investigating Transparent HugePages. An analysis of HugeTLB on the Arndale board will come at a later time.

## 3.2 Hardware setup

We ran all experiments on an Arndale board [7] Version 2.0 (1.7GHz, 32 KB L1 data cache and 32 KB L1 instruction cache, and a 1MB L2 cache). The system has 2 GB LPDDR3 RAM. Due to overheating issues, we capped the frequency at 1.2 GHz. For all the benchmarks presented here, the board did not have a heatsink, was running with the `performance` governor, and had no throttling. The board uses a rootfs served over NFS. The connection to the server is made via 100 Mbps Ethernet, through a switch shared with a number of other devices, including the two pandaboards used for THP benchmarking.

### 3.2.1 Limitations

One of Arndale board's limitations is overheating. Running any benchmark on one core alone can drive the temperature above the critical threshold. This happens on the stock board. A solution is to add a passive heatsink. Another one is to enable throttling in order to keep the board at about 85° Celsius. Without either, the board can easily reach temperatures of 90-100° Celsius.

During the usage of the Arndale board, a yet-to-be-solved board instability has been observed. Hardware issues are suspected.

Another limitation of the setup is the fact that the board only has 2 GB RAM. Given its 512-entry TLB, and 2 MB hugepages, we can hold in TLB references that cover half its memory region. Therefore it would be more interesting to see how performance is affected when the system has more RAM.

## 3.3 Benchmarks

We ran on the Arndale board the same sets of benchmarks as on the pandaboard, as described in the Benchmarks 2.3 section.

### 3.3.1 Setup

#### 3.3.1.1 Compilation flags

All benchmarks but FTQ/FWQ and STRIDE were built with the following compilation flags:

```
-O3 -fomit-frame-pointer -funroll-loops -mcpu=cortex-a15
     -mtune=cortex-a15 -mfloat-abi=hard -mfpu=neon
-funsafe-math-optimizations -ftree-vectorize -march=armv7-a
                    -DL2SIZE=1048576
```

When building FTQ/FWQ, `-O3` would actually cause the inner loop of the benchmark to be optimized away. As a result, we used flags suggested by the benchmark for other architectures (adapted for the Arndale board):

```
-O1 -ffast-math -funroll-loops -fexpensive-optimizations
       -mcpu=cortex-a15 -mtune=cortex-a15 -static
```

When building STRIDE, `-O3` would cause the inner loop of STRIDDOT and CACHEDOT to be optimized away. Therefore, we used the same compilation flags as for the other benchmarks on Strid3, Vecop and Cache, and the following flags on CACHEDOT and STRIDDOT:

```
-O1 -mcpu=cortex-a15 -mtune=cortex-a15 -mfloat-abi=hard -mfpu=neon
  -funsafe-math-optimizations -ftree-vectorize -DL2SIZE=1048576
                        -funroll-loops
```

#### 3.3.1.2 Processor affinity

For the purpose of these benchmarks, we have not set the processor affinity for any benchmark, therefore it defaults to whether the benchmark sets it or not.

#### 3.3.1.3 ATLAS

For the same reasons as with the pandaboard, described in Section 2.3.1.3: ATLAS, we chose to use ATLAS instead of BLAS for the Arndale board too. However, due to a series of issues with ATLAS, we re-used the library built specifically for the pandaboard, as described in Section 2.3.1.1: Compilation Flags.

## 3.4 Methodology

The methodology used for the Arndale board is the same as the pandaboard one, described in Section 2.5: Methodology.

## 3.5 Results

In this section we cover results of running the benchmarks on the baseline and on the THP-enabled kernel. A comparison between the two sets of results is offered in Observations 3.5.1 and in each figure's caption. Further comments comparing the Arndale and pandaboard are in Subsection 3.5.1.1: Comparison to pandaboard.

All the graphs presented in this section show the relative improvement of THP over the baseline case, with the data being normalized to the stock kernel for each benchmark individually.

For all STRIDE results, the measurements were taken when the performance stabilized for high strides or loop lengths.

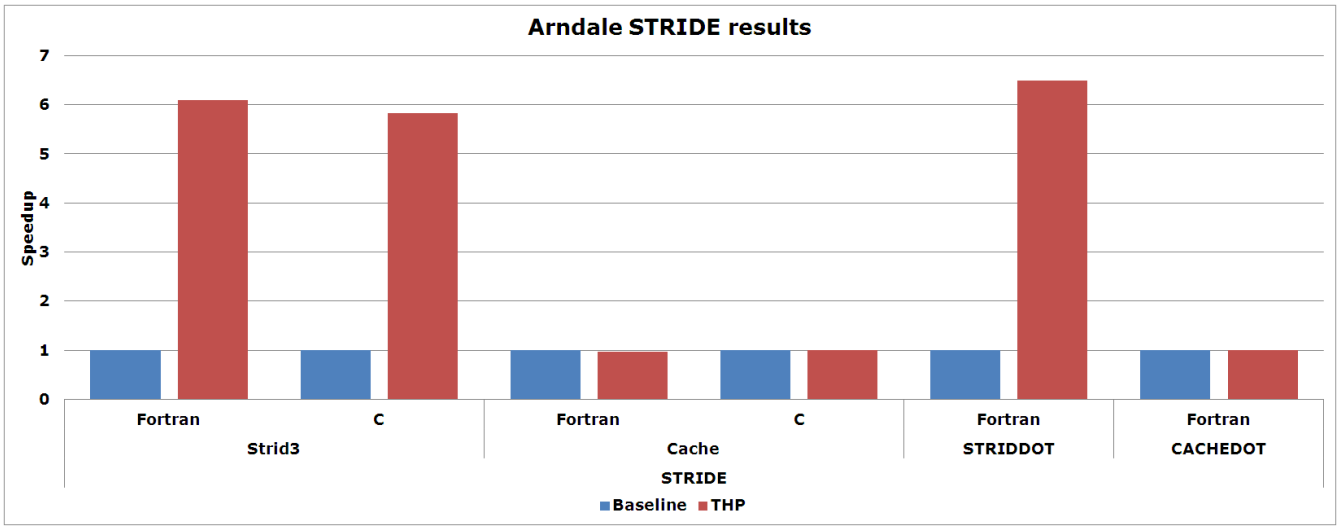


Figure 4: Here we can see the relative performance difference between the baseline and THP STRIDE Strid3, Cache, STRIDDOT and CACHEDOT. Just as for the pandaboard results shown in Figure 1, the only cases when hugepages have an effect is when we have sparse memory accesses, such as when we increase the array stride.

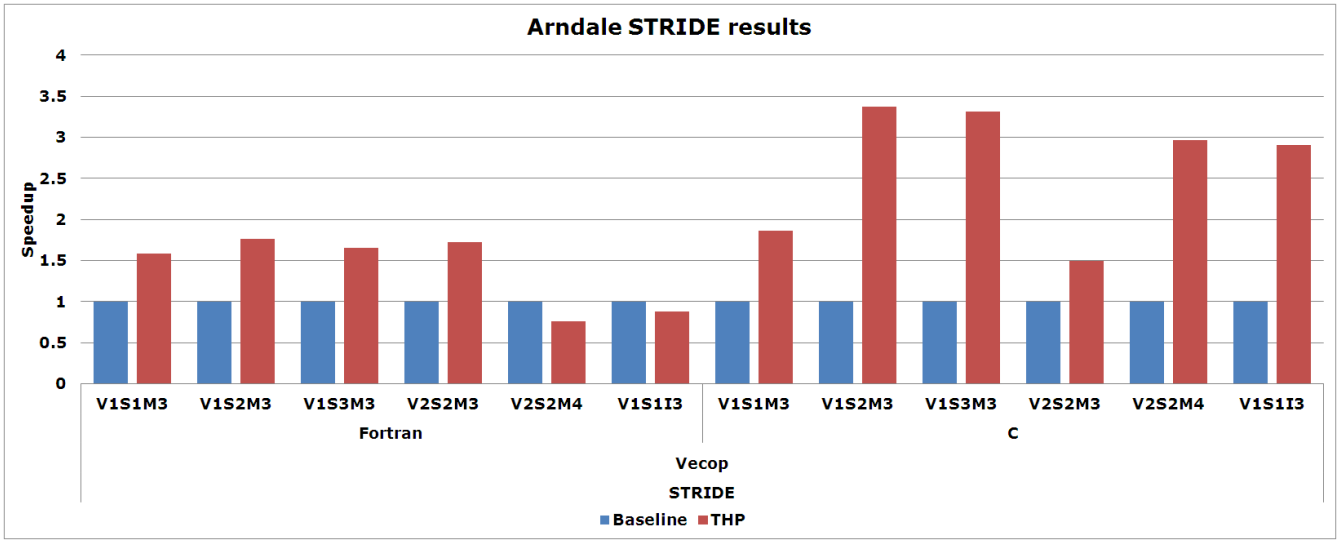


Figure 5: In this figure we can see the relative performance improvement of hugepages over the baseline case for STRIDE Vecop. These results show a significantly higher impact of hugepages on the Arndale compared to the pandaboard, as shown in Figure 2. Preliminary tests indicate the possible effects of hugepages related to cache behaviour. As these benchmarks use little data overall, the cache behaviour would have a higher impact upon their performance. When using perf for Vecop Fortran, we observed a significant decrease in L1 cache load misses when using hugepages.

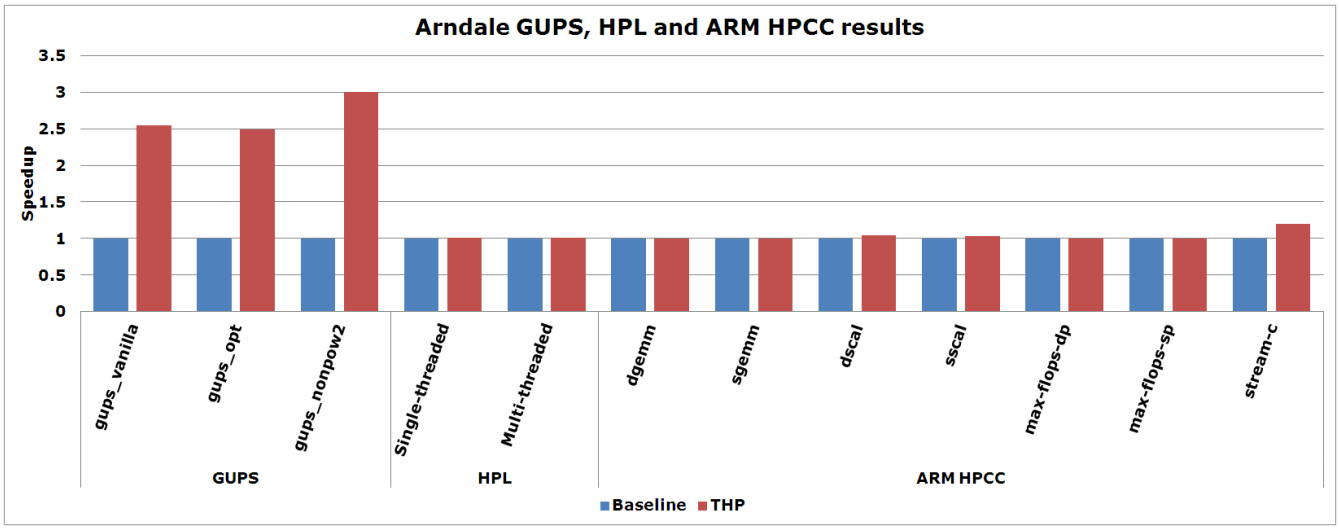


Figure 6: This figure shows us the Arndale GUPS, HPL and ARM HPCC performance difference between the baseline and THP cases. Similarly to the pandaboard case in Figure 3, hugepages made no difference on computationally bound benchmarks. The benchmarks which show no improvement are HPL and ARM HPCC, excluding stream-c. We observe an average performance improvement of about 20.31% whilst running stream-c. The highest performance benefit out of the benchmarks presented in this graph can be seen when we perform random memory accesses. GUPS presents improvements of between 2.49x and 3x.

### 3.5.1 Observations

After an analysis of all the results, we observe a trend amongst all benchmarks, which is that if benchmarks have good data locality, they will not experience a performance drop, nor gain when enabling hugepages. However, when there is little data reuse, moving towards random memory accesses, is when hugepages make a significant difference. In cases of random memory accesses, we saw improvements of up to 3x with THP, whilst in high-stride cases, up to 1024, we saw improvements of up to 6.5x.

The main observation for FWQ/FTQ benchmarks running in single-threaded mode is that there is less noise when having THP enabled. However, in multi-threaded mode, thread 0 always tends to have more noise with hugepages enabled, whilst thread 1 has less noise. Nevertheless, the differences are minor and there always is a lot more noise in the multi-threaded versions. A further investigation of why this is the case will be done at a later time, as described in Section 3.7.1.

Overall, no performance degradation has been observed whilst using Transparent HugePages on the Arndale.

A further explanation of results can be found in each figure's caption.

#### 3.5.1.1 Comparison to pandaboard

The main current observation is that on the Arndale board the effects of hugepages are more significant due to the board's higher performance dual-A15, thus the memory bandwidth limitations are reflected much better. This is shown by the higher performance increase observed when enabling THP on the Arndale, compared to the pandaboard. On the latter, the observed difference was of up to 50%, whilst on the former, of up to 6.5x. All the graphs presented in this section are normalized based on the pandaboard for each benchmark individually.

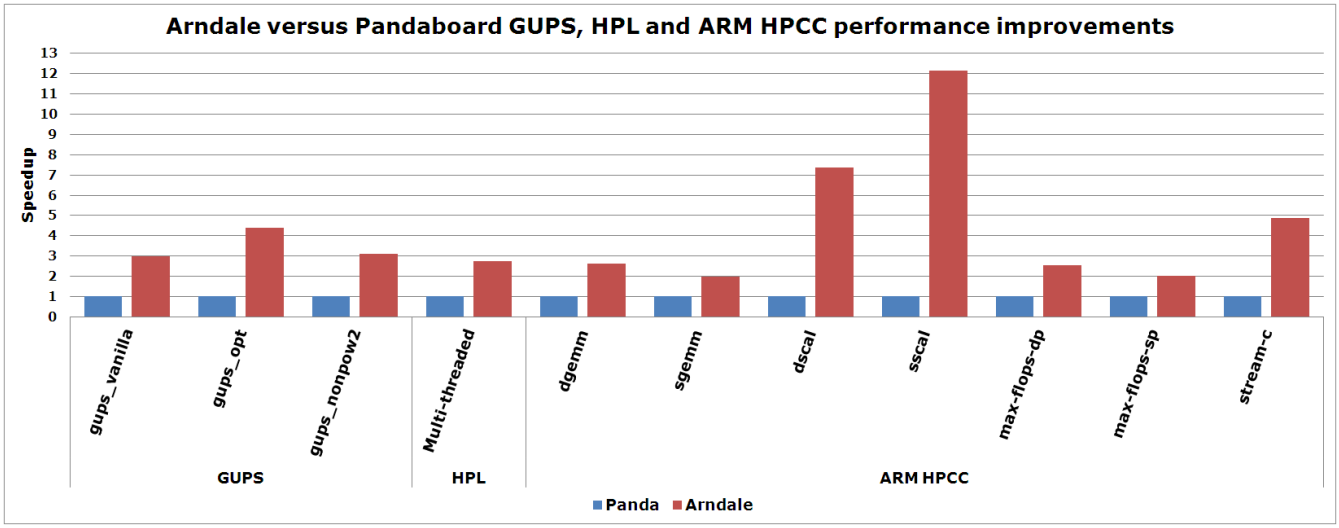


Figure 7: This figure shows us the relative performance improvement of the Arndale board over the pandaboard in the case of GUPS, HPL and ARM HPCC running using hugepages. All results are normalized based on pandaboard individually for each benchmark. In all these cases, the Arndale is more performant, the difference ranging between 1.98x to 12.16x.

As we can see in Figure 7, HPL clearly reflects the performance difference between the two boards, as the pandaboard's MPI performance is 1.028 GFlops, whilst the Arndale's is 2.837 GFlops, about 2.76x higher. This holds for both baseline and THP, as HPL did not vary its performance based on page size.

On the same figure, GUPS shows a similar improvement of about 3x higher in favour of Arndale. Comparing `gups_vanilla` when using hugepages, the pandaboard has 0.004907 GUPS, whilst the Arndale performs at 0.014691 GUPS.

For ARM HPCC, whilst comparing the `Avg. GFlops` when running with hugepages, we observed performance improvements ranging between 1.98x and 12.16x, as seen in Figure 7.

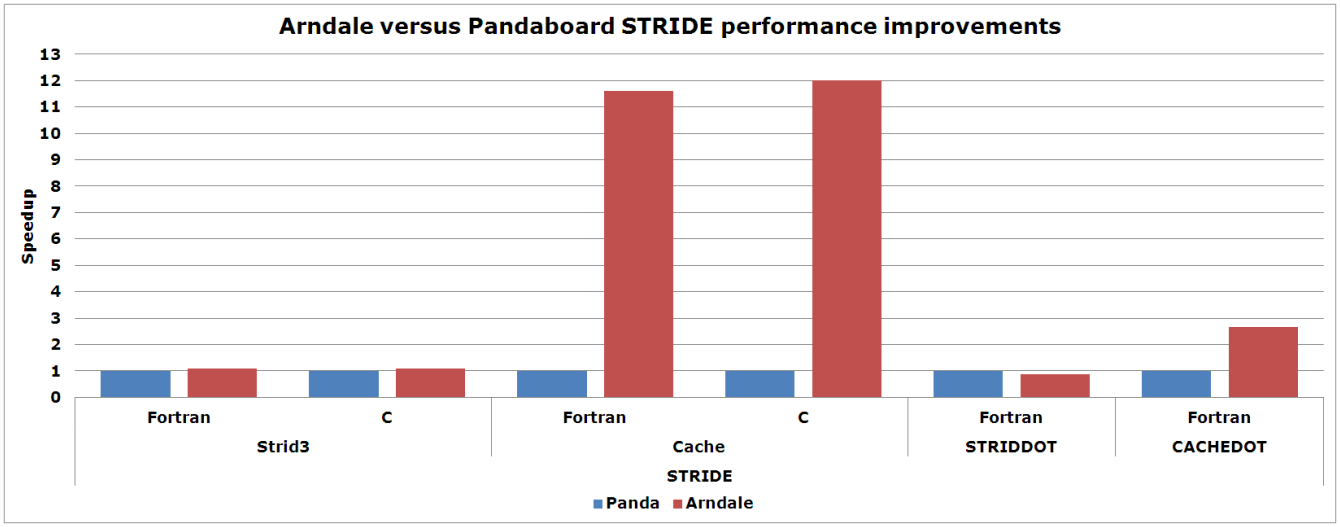


Figure 8: This figure shows us the relative performance difference between the pandaboard and the Arndale when having hugepages enabled for STRIDE Strid3, Cache, STRIDDOT and CACHEDOT.

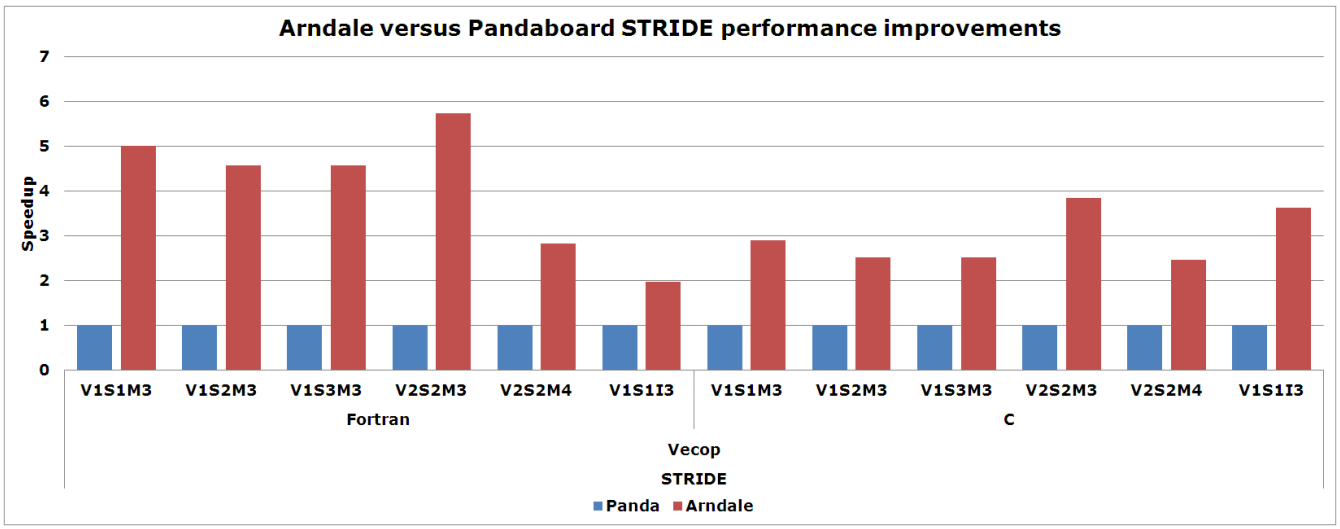


Figure 9: Here we can see the relative performance improvements of the Arndale over the pandaboard when using hugepages for STRIDE Vecop.

STRIDE is the only case when one of the constituent benchmarks had lower performance on the Arndale compared to the pandaboard. It is the Fortran STRIDDOT result presented in Figure 8, which shows the Arndale having 0.88x the performance

of the pandaboard. The comparison was made between the THP results. As we can see in Figures 8 and 9, the performance variation is between 0.88x for STRIDDOT, up to 12x for C Cache. The average performance improvement taken across all STRIDE benchmarks is of about 4x.

## 3.6 Conclusions

Just as for pandaboard, Transparent HugePages offer a good performance improvement on the Arndale, without any observable performance decrease and low kernel noise impact. As a result, we believe THP should be used for HPC purposes.

## 3.7 Further work

### 3.7.1 Kernel profiling

In order to continue optimizing the kernel, the next step is to run an in-depth analysis to fully understand the source of the noise. The approach we are taking for this is similar to the one for the pandaboard. We will make use of tools such as DS-5 with Streamline, ftrace directly and trace-cmd with kernelshark. As hardware perf events work on the Arndale, we will re-investigate the suitability of lltng (Linux Trace Toolkit - next generation).

### 3.7.2 In-depth analysis of results

As some of the benchmarks have shown similar patters of performance on both Arndale and pandaboard, we will be running a more in-depth analysis of those results as it could expose interesting and useful architectural features.

### 3.7.3 Sample HPC applications

As the synthetic benchmarks can only characterize a system to a certain extent, we will run more comprehensive experiments using sample HPC applications. We will follow-up with an in-depth analysis of the results on the Arndale to see whether the same patters of performance as with the benchmarks will be observed.

### 3.7.4 WP6 microbenchmarks

For the purpose of analysing the impact of hugepages in a variety of memory access patterns, we chose the aforementioned benchmarks. The next step is to observe and analyse the effects of THP whilst running on the WP6 microbenchmarks.

### 3.7.5 HugeTLB

If we find that it is feasible to test HugeTLB as well, we will perform a similar analysis of results for it. The main difference between THP and HugeTLB is that, in general, you

need to explicitly use the hugepages for HugeTLB, as opposed to THP, which requires changes to the benchmarks.

### 3.7.6 Interrupt routing

We will explore the effects of choosing a different routing for interrupts, other than just any core being allowed to service any interrupt. We will test having one OS-core and the other one a computational core.

### 3.7.7 Scheduling options

Currently, the kernel is using the Low-latency desktop preemption model. Another possibility is to use a Server model, which uses no forced preemption. Other scheduling possibilities, both kernel and userspace-side will be explored.

### 3.7.8 gem5 simulations

In addition to running an in-depth kernel profiling, we will use the gem5 [8] simulator to experiment with various page sizes and total amount of RAM configurations. This could also give us more detailed statistics and the ability to test on system setups we might not have easy access to.

## 3.8 Resources

Hugepage support has been added for ARM with the patch set `ARM: mm: HugeTLB + THP support.` [9].

# 4 Conclusions and Final remarks

After running all experiments on both pandaboard and Arndale, we can observe a main recurring behaviour: if there is little or no data reuse, including random memory accesses, hugepages will allow a significant performance increase, from 50% on a pandaboard, to the 6.5x observed on the Arndale. In the best case scenarios of data reuse, there was no performance increase.

Whilst running with THP enabled, we observed either no noise increase or slight noise reduction.

Overall, there was no performance degradation when running on both pandaboard, or Arndale.

As a result, due to the lack of performance decrease, the overall improvement in worse case scenarios in terms of memory accesses and the low kernel noise impact, we believe THP should be used for HPC purposes.

# References

[1] pandaboard.org. OMAP4 PandaBoard System Reference Manual. http://pandaboard.org/sites/default/files/board_reference/pandaboard-a/panda-a-manual.pdf.

[2] Lawrence Livermore National Laboratory. FTQ/FWQ, STRIDE Sequoia benchmarks source code. https://asc.llnl.gov/sequoia/benchmarks/.

[3] Steve Plimpton. GUPS Sandia source code. http://www.sandia.gov/~sjplimp/download.html.

[4] J. Dongarra A. Petitet, R. C. Whaley and A. Cleary. HPL Innovative Computing Laboratory source code. http://www.netlib.org/benchmark/hpl/hpl-2.1.tar.gz.

[5] Jack Dongarra, Piotr Luszczek, David Bailey, Jeremy Kepner, David Koester, Bob Lucas, John McCalpin, Antoine Petitet, Rolf Rabenseifner, Daisuke Takahashi, and R. Clint Whaley. HPCC source code. http://icl.cs.utk.edu/hpcc/software/index.html.

[6] Please see homepage for details. ATLAS source code. http://sourceforge.net/projects/math-atlas/files/Stable/3.10.0/atlas3.10.0.tar.bz2/download. http://math-atlas.sourceforge.net/.

[7] InSignal. Arndale 5 Base Board System Reference Manual. http://www.arndaleboard.org/wiki/downloads/supports/BaseBoard_Specification_Arndale_Ver1_0.pdf.

[8] gem5 project home page. http://www.m5sim.org/Main_Page.

[9] Steve Capper. ARM: mm: HugeTLB + THP support. http://lists.infradead.org/pipermail/linux-arm-kernel/2012-October/126382.html.