LOw Power Heterogeneous Architecture
for NExt Generation of SmaRt Infrastructure and Platforms
in Industrial and Societal Applications

D5.5 - Energy and Performance Measurements for Different Workloads Intermediate Version

Co-funded by the Horizon 2020
Framework Programme of the European Union

DELIVERABLE NUMBER	**D5.5**
DELIVERABLE TITLE	Energy and Performance Measurements for Different Workloads - Intermediate
RESPONSIBLE AUTHOR	TECHNION

GRANT AGREEMENT N.	688386
PROJECT REF. NO	H2020- 688386
PROJECT ACRONYM	OPERA
PROJECT FULL NAME	LOw Power Heterogeneous Architecture for Next Generation of SmaRt Infrastructure and Platform in Industrial and Societal Applications
STARTING DATE (DUR.)	01/12/2015
ENDING DATE	30/11/2018
PROJECT WEBSITE	www.operaproject.eu

WORKPACKAGE N. | TITLE	WP5 | Optimized Workload Management on Heterogeneous Architecture
WORKPACKAGE LEADER	IBM
DELIVERABLE N. | TITLE	D5.5 | Energy and Performance Measurements for Different Workloads - Intermediate
RESPONSIBLE AUTHOR	I.Yaniv
DATE OF DELIVERY (CONTRACTUAL)	31/05/2017
DATE OF DELIVERY (SUBMITTED)	31/05/2017
VERSION | STATUS	V1 | Final
NATURE	R(Report)
DISSEMINATION LEVEL	PU(Public)
AUTHORS (PARTNER)	TECH

VERSION	MODIFICATION(S)	DATE	AUTHOR(S)
0.1	First Draft	01/04/2017	M.Agbarya
0.9	Ready for review	15/05/2017	M.Agbarya
1.0	Modify according to IBM and TESEO reviews	26/05/2017	Mike Rapoport (IBM) Roberto Peveri (TESEO)

PARTICIPANTS		CONTACT
STMICROELECTRONICS SRL		Giulio Urlini Email: Giulio.urlini@st.com
IBM ISRAEL SCIENCE AND TECHNOLOGY LTD		Joel Nider Email: joeln@il.ibm.com
HEWLETT PACKARD CENTRE DE COMPETENCES (FRANCE)		Gallig Renaud Email: gallig.renaud@hp.com
NALLATECH LTD		Craig Petrie Email: c.petrie@nallatech.com
ISTITUTO SUPERIORE MARIO BOELLA		Olivier Terzo Email: terzo@ismb.it
TECHNION ISRAEL INSTITUTE OF TECHNOLOGY		Dan Tsafrir Email: dan@cs.technion.ac.il
CSI PIEMONTE		Vittorio Vallero Email: Vittorio.vallero@csi.it
NEAVIA TECHNOLOGIES		Jean Hubert Wilbrod Email: jean-hubert-wilbrod@neavia.com
CERIOS GREEN BV		Frank Verhagen Email: frank.verhagen@certios.nl
TESEO SPA		Stefano Serra Email: s.serra@teseo.clemessy.com
DEPARTEMENT DE L'ISERE		Olivier Latouille Email: olivier.latouille@isere.fr

ACRONYMS LIST

TLB	Translation Lookaside Buffer
THP	Transparent Huge Pages
GUPS	Giga Updates Per Second
PMH	Page Miss Handler
PMU	Processor Monitor Unit
RAPL	Running Average Power Limit

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

Work Package 5 tries to optimize the performance and energy-consumption of workloads that are running on data-centers. Specifically, task 5.5 tries to develop a model for efficient allocation of compute resources among the applications. For example, some applications may run significantly faster when they are given more threads. Another example is that allocating more huge pages to back the allocated memory may reduce the TLB miss rate and the overhead of serving TLB misses.

Optimizing the allocation of resources requires building a model that estimates and predicts the runtime overhead of different resource allocations. In this work, we will focus on two compute resources: huge pages and threads.

The aim of the task is to measure energy and performance for different data-centers workloads that run on modern, and complex CPUs (superscalar, pipelined, and out-of-order) and to build a model that can be used to estimate the correlation between the allocated compute resources, and performance (expressed by CPU time/cycles) and energy consumption.

For huge pages resource, we will develop new toolset that can control the number of huge pages allocated per application, analysing the runtime and energy-consumption overhead, and then build the model.

This document is for the intermediate deliverable, and it will describe the model for the first compute resource (huge pages) only. Specifically, we will examine the previously proposed model of runtime.

The final deliverable will also describe the study concern with threads compute resource.

Position of the deliverable in the whole project context

Work Package 5---Optimized Workload Management on Heterogeneous Architecture---deals with innovative methods for reducing energy consumption and increasing efficiency by scheduling tasks to appropriate compute resources within a data-center. Efficient resource allocation requires good understanding of how application performance is impacted. This deliverable is a research work which its goal is to study some workloads energy and performance curves, and to try to conclude a model for runtime/energy overhead when using different amount of different compute resources. Accurate and fast methods for performance estimation will be beneficial, for example, in workload management and dynamic allocation of resources, as needed in other tasks of WP5 under the OPERA project, e.g., workload management and dynamic allocation of resources in tasks: T5.2, T5.3, and T5.4.

Description of the deliverable

Task 5.5---Energy and Performance Measurements for Different Workloads---goal is to measure energy and performance for different memory-intensive workloads that run on modern, and complex CPUs (superscalar, pipelined, and out-of-order), to study the experiments results' curves, and to try to conclude a model for runtime/energy overhead when using different amount of different compute resources (this work will focus on two kinds of compute resources, huge pages and threads). Finding the correlations between using different compute resources and runtime, will help in developing a methodology for performance estimation, in terms of runtime and energy consumption.

List of actions and roles

LIST OF ACTIONS			
ACTIVITIES LIST AND PARTNERS ROLES	Technion	IBM	TESEO
Summary of the initial requirement	P		
Developing the Huge Pages Allocator toolset	P		
Studying the performance and energy consumption of GUPS benchmarks	P		
Contribution to the deliverable organization, writing, and reviewing	P	R	R

- P = Participating (includes I & R)
- I = Input delivery (Includes R)
- R = review

TABLE OF CONTENTS

1 INTRODUCTION

Work Package 5---Optimized Workload Management on Heterogeneous Architecture---deals with innovative methods for reducing energy consumption and increasing efficiency by scheduling tasks to appropriate compute resources within a data-center. Efficient resource allocation requires good understanding of how application performance is impacted. For this work, we will use the application CPU runtime for the performance estimations, and by application CPU runtime we mean the time that the CPU spent in processing the application we measure.

Task 5.5---Energy and Performance Measurements for Different Workloads---goal is to measure energy and performance for different memory-intensive workloads that run on modern, and complex CPUs (superscalar, pipelined, and out-of-order), to study the experiments results' curves, and to try to conclude a model for runtime/energy overhead when using different amount of different compute resources (this work will focus on two kinds of compute resources, huge pages and threads). Finding the correlations between using different compute resources and runtime, will help in developing a methodology for performance estimation, in terms of runtime and energy consumption.

Accurate and fast methods for performance estimation will be beneficial, for example, in workload management and dynamic allocation of resources, as needed in other tasks of WP5 under the OPERA project, e.g., workload management and dynamic allocation of resources in tasks: T5.2, T5.3, and T5.4. As these two compute resources we are studying are limited (number of huge pages that can be allocated in a system is limited and the number of threads is limited in increasing performance perspective, otherwise a large number of threads will be used by all applications that are running on multi-core systems), then allocating these limited resources between applications that run on the same system such as the performance will increase requires to have some model to find the best allocation of these resources to get the best performance.

In modern computing platforms and data-centers, the RAM size is not the main performance bottleneck, and modern computing platforms can support terabytes of RAM. But, increasing only the RAM size does not increase address translation table (TLB). Because TLB capacities cannot scale at the same rate as DRAM, TLB misses and address translation can incur crippling performance penalties for large memory workloads. TLB misses might degrade performance substantially and might account for up to 50% of the application runtime [3]. So, increasing only RAM size will not improve the performance for some applications (specially memory intensive ones), that suffer from TLB misses, for example, image processing application from the Truck use case under OPERA project. Therefore, using huge pages can save some of these penalties by the fact that using TLB entries of huge pages covers much more memory space than when using the same number of TLB entries of base pages.

Optimizing the allocation of resources requires building a model that estimates and predicts the runtime overhead of different resource allocations. We will focus on two compute resources: huge pages and threads. For building the estimation model, a profiling work should be done on these different compute resources with different workloads. There are two main challenges in running the profiling work on modern computing platforms, the first is that these platforms are designed to run multi tasks on multi cores, and then profiling work should profile only the running work without being affected by other tasks that run on the same system, or on the same core. And the second challenge, is that developing an estimation model requires few different samples of runtime for different page-walks or threads, but getting diversity in TLB page walks for the same workload is more challenging, in terms of controlling the

page walks or the allocated huge pages. In order to answer these two challenges, we developed a toolset that runs workloads in an isolated setup, by using perf, rapl, numactl, taskset, and isolcpus properties and utilities. Also, under the same toolset, we developed a new library, HugePagesAllocator, that allocates different huge pages per memory allocation transparently for flexibility control of allocated huge pages.

Our work will concentrate on memory-intensive workloads and we will choose the benchmarks set later on. But, for now we use the synthetic benchmarks set “GUPS” that measures the rate of integer random updates of memory. And, as mentioned before, in this intermediate deliverable, we will focus only on huge pages resource, and the study of threads compute resource will be delivered in the final deliverable.

2 INDICATORS

We are running the intensive-memory workloads using different number of huge pages (of 2MB in x86-64) and super pages (of 1GB in x86-64), gathering performance counters provided by the CPU (in our case, Intel CPU Performance Counters, while other vendors provide equivalent performance counters as well) using Linux perf utility, and measuring energy consumption using Intel RAPL framework in Linux. After gathering all these results and analysing them we will try to develop a model of performance estimation based on number of used huge pages.

2.1 PERFORMANCE COUNTERS

“Hardware performance counters are set of special-purpose registers built into modern microprocessors to store the counts of hardware-related activities within computer systems. Advanced users often rely on those counters to conduct low-level performance analysis or tuning. Compared to software profilers, hardware counters provide low-overhead access to a wealth of detailed performance information related to CPU's functional units, caches and main memory etc. Another benefit of using them is that no source code modifications are needed in general”. [2]

2.1.1 DTLB-LOAD/STORE-PAGE-WALK CYCLE

The main counters and hardware events we use in our work, are the DTLB_LOAD_MISSES:WALK_DURATION and DTLB_STORE_MISSES:WALK_DURATION, which count the total cycles the Page Miss Handler (PMH) is busy with page walks for memory load and store accesses, accordingly. For the GUPS benchmark we use only DTLB_LOAD_MISSES:WALK_DURATION and not the DTLB_STORE_MISSES:WALK_DURATION, because the number of memory store accesses is negligible compared with load accesses in these benchmarks (they are memory reading intensive workloads).

2.2 PERF TOOL

Perf tool is a performance analysing tool in Linux, available from Linux kernel version 2.6.31. The user-space controlling utility, named perf, is accessed from the command line and provides a number of subcommands; it is capable of statistical profiling of the entire system (both kernel and user-land code). It supports hardware performance counters, tracepoints, software performance counters, and dynamic probes. For our work, we use perf to monitor hardware performance counters of CPU cycles, energy consumption, and page-walk cycles. [4, 5]

2.2.1 Page Walk Events

The main counters and hardware events we use in our work, are: the DTLB_LOAD_MISSES:WALK_DURATION and DTLB_STORE_MISSES:WALK_DURATION, which count the total cycles PMH is busy with page walks for memory load and store accesses, accordingly. For the GUPS benchmark we use only DTLB_LOAD_MISSES:WALK_DURATION and not the DTLB_STORE_MISSES:WALK_DURATION, because the number of memory store accesses is negligible compared with load accesses in these benchmarks (they are memory reading intensive workloads).

Perf tool does not have built-in events for these hardware counters, but it supports monitoring events according to their raw Processor Monitor Unit (PMU) descriptors. The raw PMU event is in the form of rNNN where NNN is a hexadecimal event descriptor, and these descriptors are CPU specific. In our work, we support various processors by using libpfm4 library for getting the page-walk-duration event descriptor of the specific running processor. [6]

2.2.2 Energy Events

The toolset, which we provide, collects energy events as well, besides the page-walk-duration events. Currently, the toolset supports collecting energy events only when running on Intel processors, because we use Intel® RAPL interface.

2.3 LIBPFM4

Using perf tool to monitor hardware performance events, requires to provide the perf tool the list of hardware events codes that should be monitored. But, these events codes are CPU specific, and if our tool will be run in different machines with different CPUs, then the list of events should be adapted to match the correct codes of the target machine CPU, and that requires our tool to maintain a mapping for all available processors and all their hardware events. For getting the hardware events codes, we use the free library libpfm4, which is a helper library to help encode Performance Events to use with perf tool. Currently, libpfm4 support Hardware PMU for X86-processors, and according to libpfm4 documentation, the support for non x86-processors will be added gradually. [7]

2.4 RAPL

Starting from Intel® Sandy Bridge processors, Intel introduced the *Running Average Power Limit (RAPL)* interface for exposing power meters and power limits.

RAPL provides a set of counters providing energy and power consumption information. RAPL is not an analog power meter, but rather uses a software power model. This software power model estimates energy usage by using hardware performance counters and I/O models. Based on Intel measurements, they match actual power measurements. [8, 9]

There are, currently, three available ways to read RAPL counters:

1- Using raw-access to the underlying MSRs under /dev/msr.
2- Reading the files under /sys/class/powercap/intel-rapl/intel-rapl:<socket-number>
   This was introduced in Linux 3.13
3- Using the perf tool, starting from Linux 3.14.

We use the perf tool, in our work, for reading energy counters.

3 WORKLOADS

As we mentioned in Introduction, our work will focus on memory-intensive workloads. For the intermediate deliverable we choose to use GUPS synthetic benchmark.

Giga-updates per second (GUPS) is a measure of computer performance, and how frequently a computer can issue updates to randomly generated RAM locations. GUPS measurements stress the latency and especially bandwidth capabilities of a machine.

For the next deliverable, we will choose the representative benchmarks for the different use cases on OPERA project.

4 METHODOLOGY

We run the representative workloads through few tools and applications to better control memory allocations, collect necessary data from experiments, and analyse the results.

4.1 RUNNING IN ISOLATED ENVIRONMENT (CORES AND NUMA NODES)

Multi-core systems, in the modern computing age, can be found everywhere, starting from a small handheld smartphone to the big data-centers servers. Modern computing platforms, which are multi-core systems, run modern operating systems, which support multitasking by default and can run multiple processes simultaneously. As multitasking greatly improved the throughput of computers, it become widely used by operating systems and applications. But, for scientific experiments and analysis, researchers, often, prefer to disable this feature because it can cause noises in the experiments results.

In our work, any background application, which runs in the same processor as our profiling work, may cause noises in our results due to using and filling the same cache with the background application data. Therefore, we would like to isolate the execution of our tool and benchmarks, as much as possible, from any running application or system service in the background.

Non-uniform memory access (NUMA) is another element in multi-core systems that can affect scientific research results. NUMA is a computer memory design used in multiprocessing, where the memory access time depends on the memory location relative to the processor. Under NUMA, a processor can access its own local memory faster than non-local memory (memory local to another processor or memory shared between processors). So, the runtime of running a benchmark, which allocated huge space of memory, can be affected by the way the memory is allocated across system NUMA nodes, because access time of different NUMA nodes from the same processor is vary by the distance of the NUMA node from this specific processor. Therefore, we need to do some isolation work for NUMA nodes as well.

In order to achieve the isolation, we use the following techniques:

- Isolcpus
  This option is a Linux kernel command-line parameter. And it is can be used to specify one or more CPUs to isolate from the general SMP balancing and scheduling algorithms. The only way to move a process onto or off an "isolated" CPU via the CPU affinity syscalls or cpuset.
  We use this option, in our work, to isolate a processors group to be dedicated for our work and experiments. And when we run our tool and benchmarks we bind the execution to one of the isolated processors, which was defined in the isolcpus kernel command-line option.
- Numactl
  “numactl” is a tool that runs processes with a specific NUMA scheduling or memory placement policy. The policy is set for command and inherited by all of its children. In addition it can set persistent policy for shared memory segments or files. [Linux manual pages]
  We use this tool to bind the running benchmark to some specific NUMA node, which is connected directly to the running processor we bind our benchmark to it, for isolating any possible noise in results, which may be caused by accessing huge pages from different NUMA nodes.
- Taskset
  “taskset” is a tool that is used to set or retrieve the CPU affinity of a running process given its pid, or to launch a new command with a given CPU affinity. CPU affinity is a scheduler property that "bonds" a process to a given set of CPUs on the system. The Linux scheduler will honor the given CPU affinity and the process will not run on any other CPUs. [Linux manual pages]
  We use this tool to bind the running benchmark to some specific processor, for isolating any possible noise in results, which may be caused by switching to another processor and losing beneficial of loaded data to current processor cache.

4.2 CONFIGURATION THE SYSTEM

Allocating huge pages in Linux operating system requires preserve memory space for huge pages use. This is necessary because allocating memory area backed by huge pages requires the system to find contiguous physical memory space per huge page, which can fail in fragmented system, and pre-allocating contiguous areas for huge pages can decrease the failure possibility to find enough contiguous physical memory when allocation request is sent to the kernel.

It should be mentioned that it is preferable to run our tool set on clean system after boot to work with un-fragmented system, which increases success rate of huge pages pre-allocation.

Pre-allocating and preserving huge pages can be done through hugeadm Linux utility:

- Hugeadm

  "hugeadm" is a tool that displays and configures the systems huge page pools. The size of the pools is set as a minimum and maximum threshold. The minimum value is allocated up front by the kernel and guaranteed to remain as hugepages until the pool is shrunk. If a maximum is set, the system will dynamically allocate pages if applications request more hugepages than the minimum size of the pool. There is no guarantee that more pages than this minimum pool size can be allocated. [Kernel manual pages]

  Before running any benchmark that requires huge pages allocation, our tool configures the huge pages pools to pre-allocate huge pages as required by the benchmark.

4.3 CONTROLLING PAGES SIZES FOR WORKLOAD MEMORY ALLOCATIONS

One of the main targets of our work is to get variety in results, by getting multiple different points on the chart (i.e., getting multiple different runtime and page walks cycles for the same workload), for good analysis and conclusions, and for building more accurate estimation model. For getting good variety in results, we need to get different results of the same benchmark/workload in terms of runtime/energy and walk duration, in order to investigate the overhead of page walks on runtime/energy. So, we need to control, somehow, the number of page walks for the same workload in order to get different runtime/energy and then trying to find a correlation between them.

Page walks can be controlled by controlling pages sizes of the allocated memory, and for huge pages there will be less page walks because there will be less TLB misses because the TLB entries will cover larger memory space. To control pages sizes and use huge pages, there are few available solutions in Linux for that:

1- Using Transparent Huge Pages (THP) which automatically promote memory allocations pages from base pages to huge pages dynamically and transparently.
2- Using hugetlbfs which is a bare interface to the huge page capabilities of the underlying hardware; taking advantage of it requires application awareness or library support.
3- Libhugetlbfs which makes heavy use of hugetlbfs interface when automatically backing regions with huge pages.
4- mmap system call can be sued to allocate memory backed by huge pages, but it requires applications modification to use it explicitly.

But, all of these existing solutions cannot achieve our goals, either they require application awareness and modification or they allocate all the memory backed by huge pages and do not give flexibility to fully control the number of huge and base pages for memory allocations.

We introduce new library, as a part of our toolset, which is called HugePageAllocator, to control memory allocations for existing applications transparently (i.e., without modifying them or even rebuilding them).

4.3.1 HugePageAllocator:

The HugePageAllocator library re-implements malloc and free APIs to control huge pages allocation for chosen allocations. Besides HugePageAllocator, there is another utility as part of the toolset that redirects malloc and free calls to HugePageAllocator library instead of libc library.

Backing memory allocations with huge pages should consider few limitations and requirements:

- It should be able to control memory allocations to be backed by different number of huge pages while keeping on their contiguity.

  Current allocators, like LibC malloc, and memory management mechanisms, like THP or hugetlbfs, do not give full control on how much of the allocated memory to be backed by huge pages and how much to be backed by base pages. E.g., malloc, always allocates memory backed by base pages, and THP or hugetlbfs either allocates all the chunk with huge pages or with base pages, none of them can allocate mixed memory with different pages.
- It should back memory allocations with huge pages without modifying applications code, i.e., it should do that transparently and should work with closed binaries. Also, it should work with any kernel (starting from version X) without modifying the kernel or requiring kernel patch.

The HugePageAllocator should fill the above requirements, and it does that by redirecting malloc and free calls, dynamically, to the overridden implementations by the library. Currently, HugePageAllocator's malloc back allocated memory by huge pages, only for chunks larger than some threshold with specific number of huge pages. For the final deliverable, it should be able to back any memory allocation by huge pages starting from allocation point until some other allocation point, and it should be applied to all allocations sizes.

5 MEASUREMENTS

To measure the overhead of page walks on runtime and energy, we do profile memory intensive workloads and plot the runtime/energy-consumption as a function of page walks duration.

The profiling work is done by running the workload several times, and in each run, our tool configures the system and the workload to work with different number of each page size.

As stated before, for the intermediate deliverable we run the workloads while backing the allocated memory with three different huge page sizes:

1- Backing all memory with base pages, and according to our assumptions, this run will have the longest runtime.
2- Backing all memory with huge pages of size 2MB.
3- Backing all memory with huge pages of size 1GB, and according to our assumptions, this run will have the shortest runtime.

5.1 SETUP DETAILS

We run our experiments and collect the measurements using the following setups. Note that GUPS three benchmarks were executed on two different platforms because we run the GUPS 2GB and GUPS 8 GB on the first platform, and then we wanted to run GUPS 32GB as well to get more results, but the first platform has no sufficient memory to run the GUPS 32GB benchmark, so we used the second platform for that. For the next deliverable we will run all benchmarks on the same platform (the second one):

5.1.1 Setup 1 – For GUPS 2GB and GUPS 8GB

- System name: Dell PowerEdge R420
- Processor: 2*Intel Xeon E5-2420 1.90GHz 6C/socket
- Sockets: 2.
  - Each Package:
  - NUMA Node Memory Size: 16GB
  - Cache L3: 15MB
  - Cores: 6
- Caches (per core):
  - L2: 256KB
  - L1d: 32KB
  - L1i: 32KB
- Memory:
  - Size: 32GB
  - Slots: 2 (16GB on each slot)
  - Speed: 1333MHz
- Operating System: Ubuntu 16.04.1 LTS

5.1.2 Setup 2 – For GUPS 32GB

- System name: Supermicro SYS-8028B-C0R3FT/X10QBL-CT
- Processor: 2*Intel(R) Xeon(R) CPU E7-4830 v3 @ 2.10GHz
- Sockets: 2.
  - Each Package:
  - NUMA Node Memory Size: 63GB
  - Cache L3: 30MB

- Cores: 12
- Caches (per core):
  - L2: 256KB
  - L1d: 32KB
  - L1i: 32KB
- Memory:
  - Size: 128GB
  - Slots: 2 (16GB on each slot)
  - Speed: 1333MHz
- Operating System: Ubuntu 14.04.5 LTS

5.2 RUNTIME GRAPHS

In this section, we present runtime graphs for GUPS benchmark with different configuration (memory allocation size). The runtime graphs show the benchmark runtime as a function of DTLB load page walks (both in terms of CPU cycles). Our initial assumption is that page walks have a linear overhead on runtime, and that what we can see in the initial graphs. However, in the analysis section we will discuss these results and next work to do.

5.2.1 Runtime As a function of Page-Walk-cycles

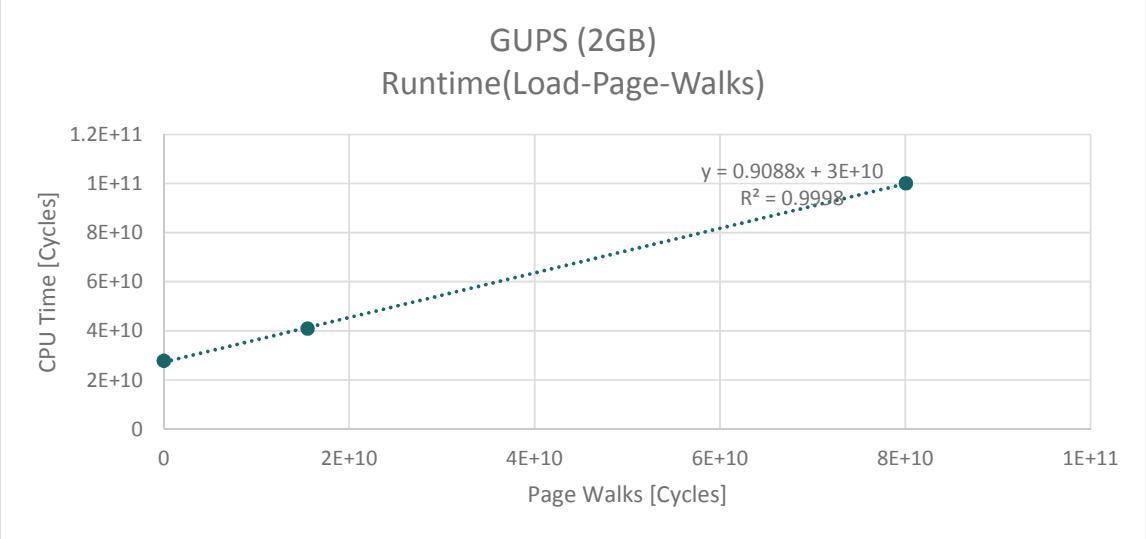

Figure 1 Runtime overhead per TLB Page Walk cycle – GUPS benchmark (2GB)

Figure 1 shows the three points of runtime and page walks when running GUPS 2GB benchmark three times: the first time while backing all the 2GB memory with base pages (the most right point), the second time while backing the memory with huge pages of size 2MB (the middle point), and the third time while backing all the memory with huge pages of size 1GB (the most left point). We used these three points and plotted a linear regression line that goes through these three points, it can be seen that the line highly explain the three points.

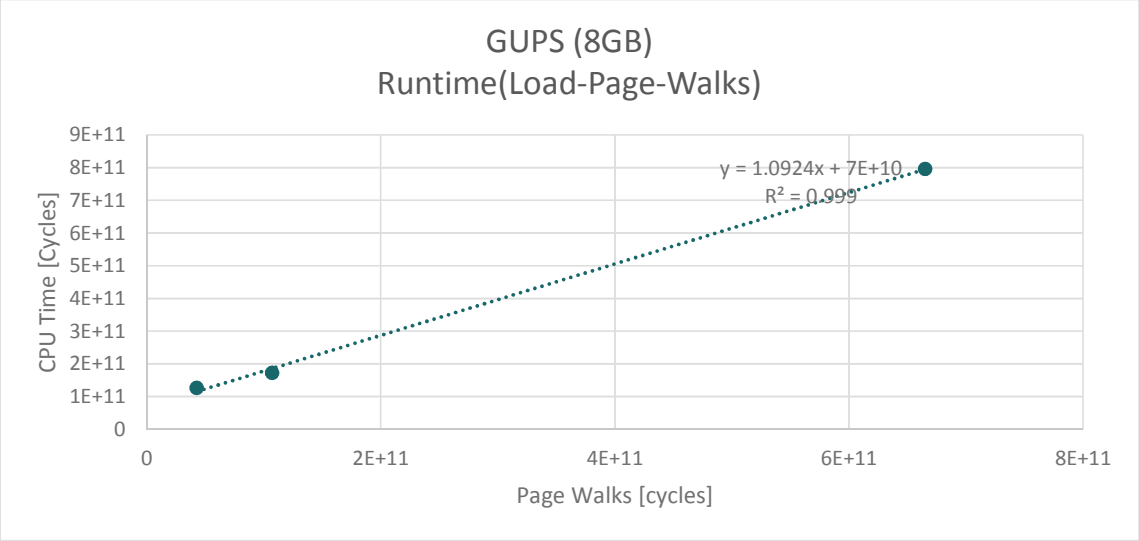

Figure 2 Runtime overhead per TLB Page Walk cycle – GUPS benchmark (8GB)

Figure 2 shows the three points of runtime and page walks when running GUPS 8GB benchmark three times: the first time while backing all the 8GB memory with base pages (the most right point), the second time while backing the memory with huge pages of size 2MB (the middle point), and the third time while backing all the memory with huge pages of size 1GB (the most left point). We used these three points and plotted a linear regression line that goes through these three points, it can be seen that the line highly explain the three points.

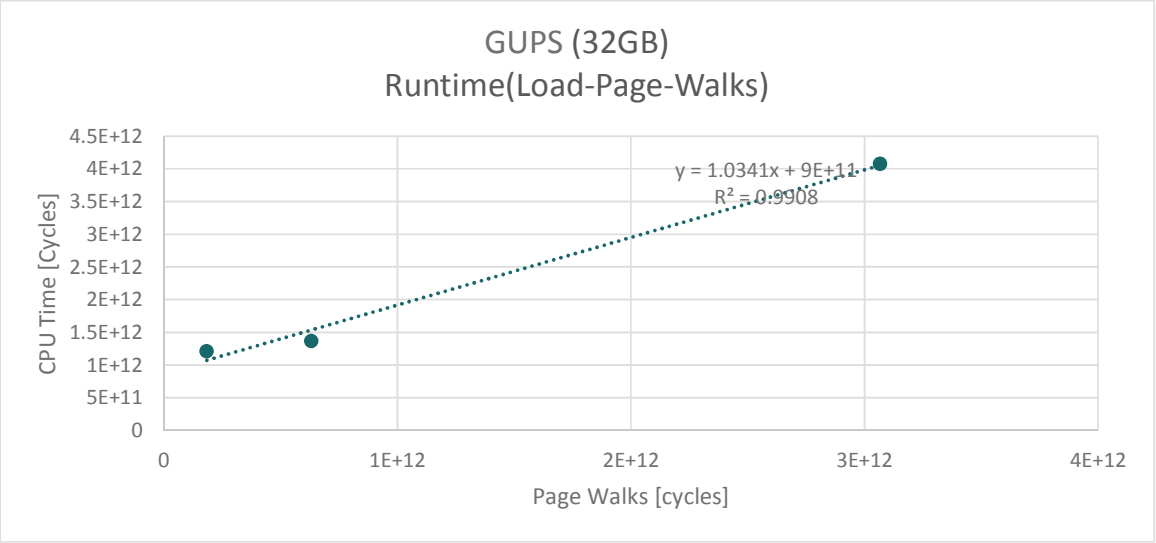

Figure 3 Runtime overhead per TLB Page Walk cycle – GUPS benchmark (32GB)

Figure 3 shows the three points of runtime and page walks when running GUPS 32GB benchmark three times: the first time while backing all the 32GB memory with base pages (the most right point), the second time while backing the memory with huge pages of size 2MB (the middle point), and the third time while backing all the memory with huge pages of size 1GB (the most left point). We used these three points and plotted a linear regression line that goes through these three points, it can be seen that the line highly explain the third point (the most right), but not like the other two points (it can be seen that the two points at the right are not fallen exactly on the regression line).

5.2.2 Runtime As a function of huge pages number

In the next deliverable, the chart will include much more points of runtime as a function of page-walks by using our toolset, and then we could plot a chart that describes how the runtime is affected by the used number of huge pages in memory allocations.

5.3 ENERGY GRAPHS

In current work, we also interested in investigating the page walks overhead on energy as well, besides the overhead on runtime. And our initial assumption is that page walks has the same overhead on energy as runtime, i.e. the overhead will be linearly as well.

In the energy graphs we present the energy consumption (in Joules) as a function of DTLB load page walks (in CPU cycles) for three flavours of the Random Access benchmark of GUPS.

5.3.1 Energy As a function of Page-Walk-cycles

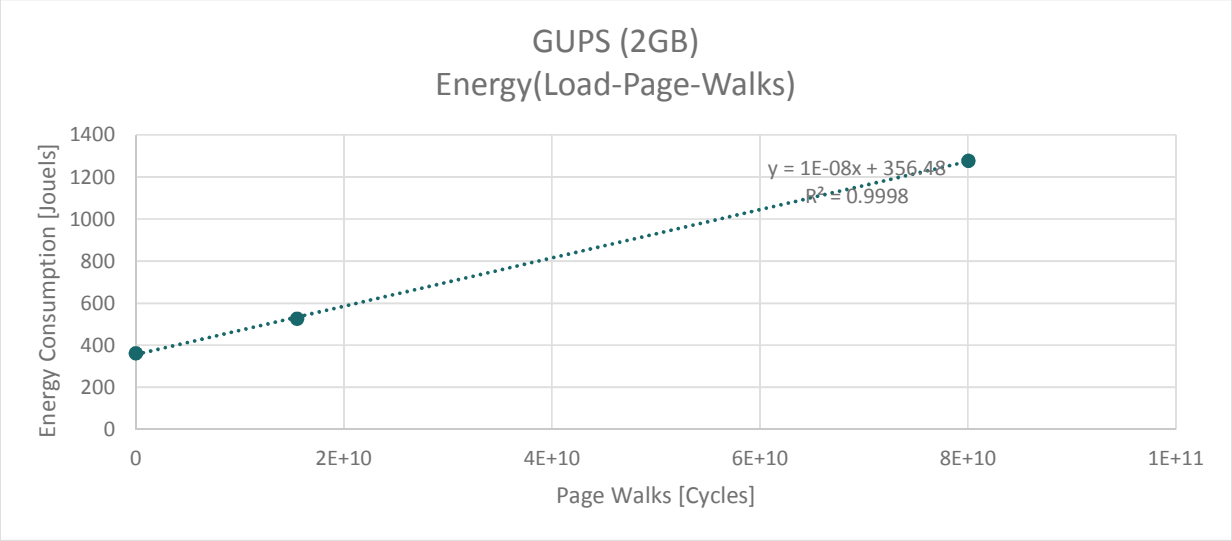

Figure 4 Energy consumption overhead per TLB Page Walk cycle – GUPS benchmark (2GB)

Figure 4 shows the three points of energy consumption and page walks when running GUPS 2GB benchmark three times: the first time while backing all the 2GB memory with base pages (the most right point), the second time while backing the memory with huge pages of size 2MB (the middle point), and the third time while backing all the memory with huge pages of size 1GB (the most left point). We used these three points and plotted a linear regression line that goes through these three points, it can be seen that the line highly explain the three points.

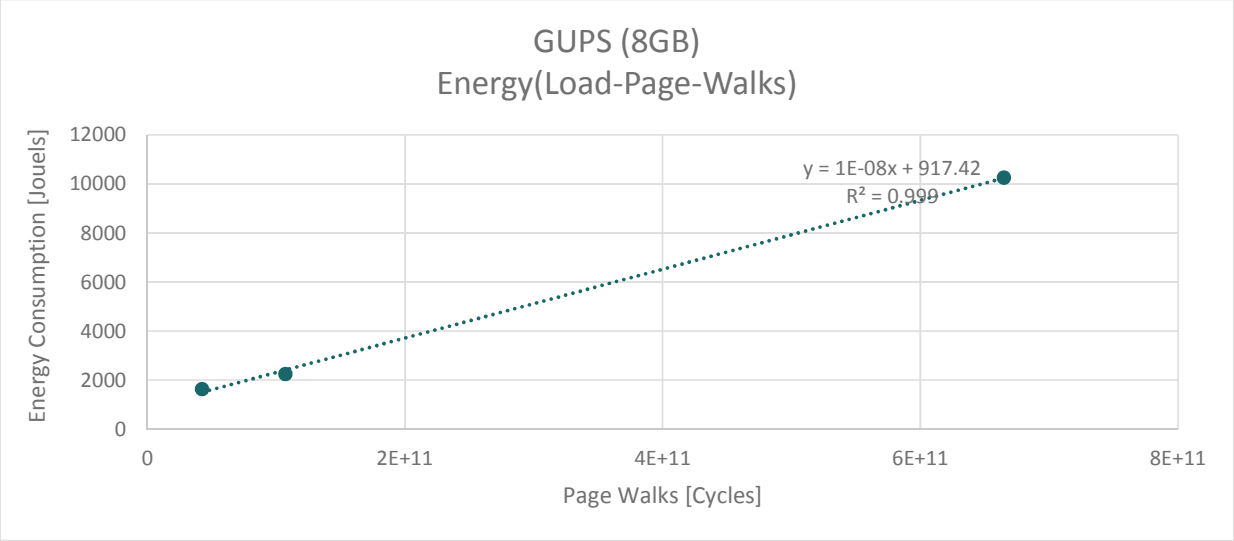

Figure 5 Energy consumption overhead per TLB Page Walk cycle – GUPS benchmark (8GB)

Figure 5 shows the three points of energy consumption and page walks when running GUPS 8GB benchmark three times: the first time while backing all the 8GB memory with base pages (the most right point), the second time while backing the memory with huge pages of size 2MB (the middle point), and the third time while backing all the memory with huge pages of size 1GB (the most left point). We used these three points and plotted a linear regression line that goes through these three points, it can be seen that the line highly explain the three points.

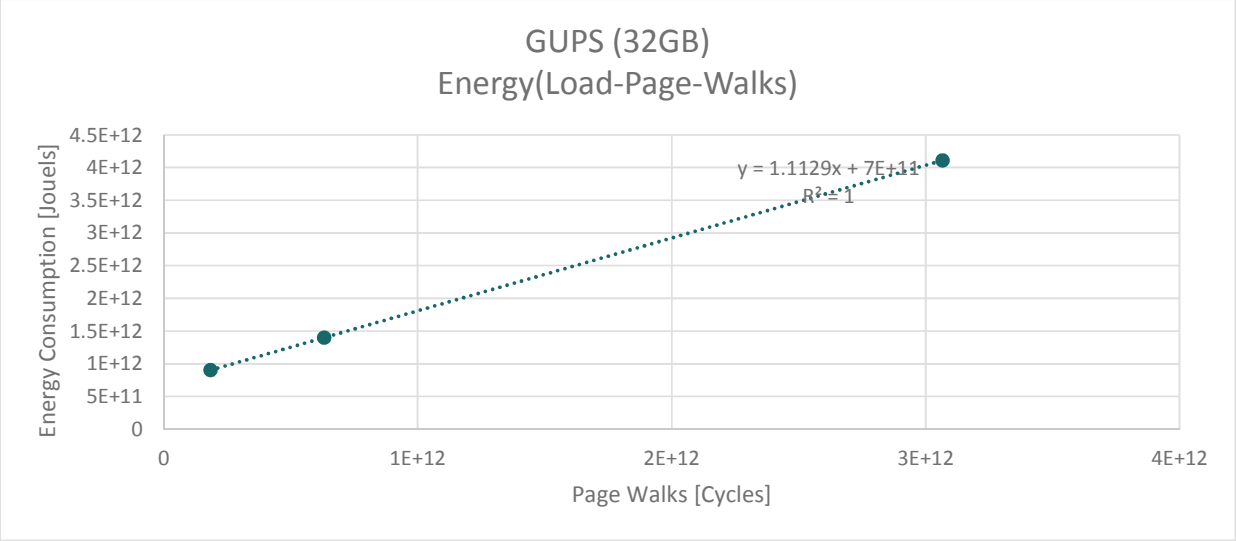

Figure 6 Energy consumption overhead per TLB Page Walk cycle – GUPS benchmark (32GB)

Figure 6 shows the three points of energy consumption and page walks when running GUPS 32GB benchmark three times: the first time while backing all the 32GB memory with base pages (the most right point), the second time while backing the memory with huge pages of size 2MB (the middle point), and the third time while backing all the memory with huge pages of size 1GB (the most left point). We used these three points and plotted a linear regression line that goes through these three points, it can be seen that the line highly explain the three points.

5.3.2 Energy As a function of huge pages number

In the next deliverable, the chart will include much more points of energy as a function of page-walks by using our toolset, and then we could plot a chart that describes how the energy consumption is affected by the used number of huge pages in memory allocations.

6 ANALYSIS

Examining the previously proposed model of runtime shows that it was developed by measuring two samples of page walks and their runtime and assuming that page walks overhead on runtime behaves linearly. The two samples were measured by running workloads backing all their memory allocations with huge pages (of size 2MB) by enabling the THP, and running them again while backing the memory allocations with base pages by disabling the THP. The model was developed by calculating the linear formula for these two samples, and then the new formula will be used to estimate runtime overhead for different page walks.

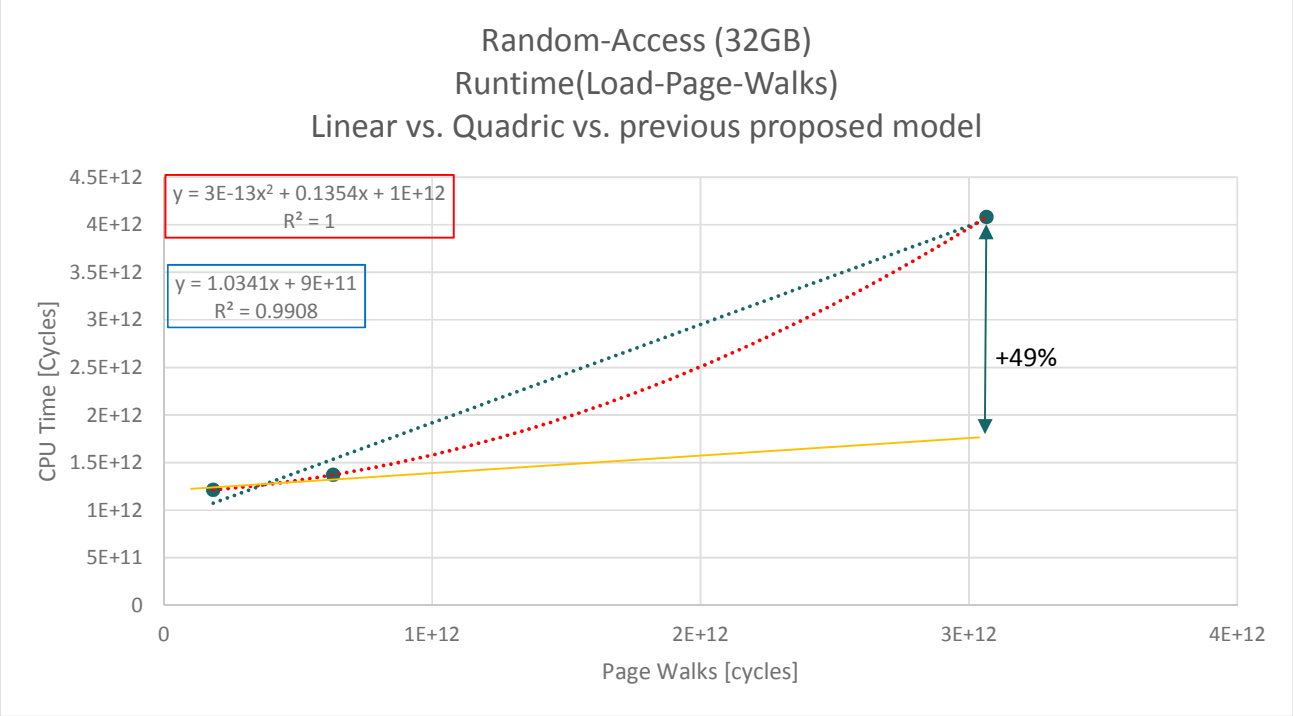

Figure 7 Runtime overhead per TLB Page Walk cycle – GUPS benchmark (32GB). Comparison Chart

As our work introduced new sample, by measuring runtime when backing all allocated memory with huge pages of size 1GB by using our toolset, a new problem introduced: which two samples should be taken to build the linear formula? As it can be seen in Figure 7, there are three samples took by running the GUPS (32GB) workload three times: the first time backing all the 32GB memory allocation with base pages (the most right point), backing the memory with huge pages of size 2MB (the middle point), and backing all the memory with huge pages of size 1GB (the most left point). Drawing a regression linear line gives the dotted blue line with R^2=0.99, while drawing a polynomial line of 2nd order gives dotted red line with R^2=1, and drawing a linear line that goes through only the two points on the left (by using the previously proposed model of two points) gives the yellow line. We get error up to +49% (as it shown in Figure 7) between the three points model (the blue line) and the previously proposed model of two points (the yellow line), and here is a full comparison between the real runtime value of the three points compared to the estimated runtime by using previously proposed model (the two points model). First and second samples in the table shows which two points were selected to build the two points' model and the error corresponds to the difference between the third point real runtime and estimated one:

Page size used to back memory allocation of first sample	Page size used to back memory allocation of second sample	Error
2MB	1GB	49%
4KB	1GB	18%
4KB	2MB	42%

Table 1 Error percentage of the difference between the real runtime and estimated runtime using previously proposed model (of two points)

For some workloads (e.g., GUPS 32GB), using quadratic model (polynomial line of second order) seems to be more accurate than linear model ($R^2 = 1$ for the quadratic model, where it is less than 1 for the linear model). While for the other benchmarks, the linear model seems sufficient.

The R-squared (R^2) we used to determine which model is better is a statistical measure of how close the data are to the fitted regression line. $R^2=0$ indicates that the model explains none of the variability of the response data around its mean, and $R^2=1$ indicates that the model explains all the variability of the response data around its mean.

Also, it can be seen through figures 4, 5, and 6 that using only three points, to express the energy consumption model, gives that the linear model is sufficient; all the three charts have $R^2>0.99$.

There is a wide range without measurements that should be covered to get reliable and accurate model. Assuming that the runtime model behaves linearly by sampling only two or three points is not sufficient, and could lead to the errors shown in previous table and figures. Therefore, our tool set will allow to mix different page sizes, for the same memory allocation, and, as a result, collect more samples.

For the intermediate deliverable, we used GUPS benchmarks, which were used to measure runtime overhead, that access the memory in a specific pattern. But, for the final deliverable, we will examine other workloads with non-unique access patterns.

7 CONCLUSIONS

Developing a reliable model for runtime overhead, caused by page walks, by sampling only two or three different points is not sufficient and could not conclude the mathematical model of the runtime overhead, i.e., if the runtime overhead behaves linearly, polynomial, etc. Assuming the linearity of the model by using only two samples could lead to a major error as we showed that error of estimated runtime could get up to 49% of the real runtime. Therefore, a wider range should be measured and sampled for building a reliable model.

REFERENCES

[1] Transparent hugepage support. https://www.kernel. org/doc/Documentation/vm/transhuge.txt, 2017. Linux documentation page (Accessed: Apr 2017).

[2] Hardware performance counter. https://en.wikipedia.org/wiki/Hardware_performance_counter, 2017. Wikipedia (Accesses: Apr 2017)

[3] I. Yaniv, D. Tsafrir; Hash, Don't Cache (the page table). In ACM Special Interest Group (SIG) for the computer systems performance evaluation community (SEGMETRICS) 2016.

http://dx.doi.org/10.1145/2896377.2901456

[4] Perf: Linux profiling with performance counters. https://perf.wiki.kernel.org/index.php/Main_Page, 2017. Perf Wiki (Accesses: Apr 2017)

[5] Perf (Linux). https://en.wikipedia.org/wiki/Perf_(Linux), 2017. Wikipedia (Access: Apr 2017).

[6] Linux Manual Pages - Perf Stat. http://man7.org/linux/man-pages/man1/perf-stat.1.html, 2017. Linux man-pages (Accesses: Apr 2017)

[7] The hardware-based performance monitoring interface for Linux, http://perfmon2.sourceforge.net/manv4/libpfm.html, 2017. Perfmon Open-Source (Accesses: Apr 2017)

[8] Reading RAPL energy measurements from Linux. http://web.eece.maine.edu/~vweaver/projects/rapl/ , 2017 (Accesses: Apr 2017)

[9] RUNNING AVERAGE POWER LIMIT – RAPL. https://01.org/blogs/2014/running-average-power-limit-%E2%80%93-rapl, 2017. Intel Open Source (Accesses: Apr 2017)