ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA019

OPTIMIZED CALCULATION OF TIMING FOR PARALLEL BEAM OPERATION AT THE FAIR ACCELERATOR COMPLEX

A. Schaller, J. Fitzek, GSI, Darmstadt, Germany
F. Wolf, D. Lorenz, TU Darmstadt, Germany

Abstract

For the new FAIR accelerator complex at GSI the settings management system LSA is used. It is developed in collaboration with CERN and until now it is executed strictly serial. Nowadays the performance gain of single core processors have nearly stagnated and multicore processors dominate the market. This evolution forces software projects to make use of the parallel hardware to increase their performance. In this thesis LSA is analyzed and parallelized using different parallelization patterns like task and loop parallelization. The most common case of user interaction is to change specific settings so that the accelerator performs at its best. For each changed setting, LSA needs to calculate all child settings of the parameter hierarchy. To maximize the speedup of the calculations, they are also optimized sequentially. The used data structures and algorithms are reviewed to ensure minimal resource usage and maximal compatibility with parallel execution. The overall goal of this thesis is to speed up the calculations so that the results can be shown in an user interface with nearly no noticeable latency.

MOTIVATION

To allow the commissioning and operation of the Facility for Antiproton and Ion Research (FAIR), the software used today has to be optimized. The CRYRING (YR), with its local injector, acts as a test facility for the new control system and in particular for the control systems' central component, the settings management system LHC Software Architecture (LSA) [1]. For the last YR commissioning beam time, about 3 700 manual trims (modifications of settings in LSA, that leads to a recalculation of all dependent parameter settings) were calculated per week with 80 working hours, which is about one trim every 77 seconds. Since the YR is a rather small accelerator ring, with a circumference of approximately 54 m [2], everything worked fine. The waiting time for these 3 700 trims summarizes to about 19 minutes distributed over the 80 working hours. The human reaction time is not much less, so the settings management system is pleasant to operate. But when it comes to calculate the Heavy Ion Synchrotron 18 (SIS18) or SIS100, the calculations get very slow. To calculate 3 700 trims for the SIS18, with its approximate 216 m [3], an operator would have to wait for over 13 hours distributed over the 80 working hours. The SIS100, with approximately 1100 m [4], calculation would even take over 91 hours which even doesn't fit into the 80 working hours. In 2025 not only single ring accelerators will be calculated separately, but the entire accelerator complex since the individual rings influence each other. Beams will start in the Universal Linear Accelerator (Unilac) and go their ways through the different rings and transfer lines to reach a target at the end, see also Figure 1. With the current calculation times it won't be possible to support an efficient beam operation for FAIR.

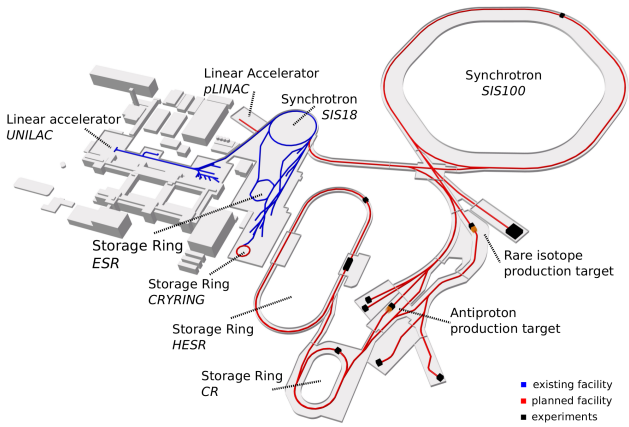


Figure 1: The layout of FAIR. The existing GSI Helmholtz Center for Heavy Ion Research (GSI) facility (blue) acts as injector for the new FAIR facility (red).

SPEEDUP

The speedup S represents a factor, that shows how two different algorithms perform on the same task. In the context of parallelization, the "speedup is a multiplier indicating how many times faster the parallel program is than its sequential counterpart. It is given by

$$S(P) = \frac{T(1)}{T(P)} \tag{1}$$

where $T(n)$ is the total execution time on a system with n processing units" [5]. $T(1)$ is also representable as

$$T(1) = T_{\text{setup}} + T_{\text{compute}} + T_{\text{finalize}} \tag{2}$$

Since the only part that can benefit from parallel optimization is T_{compute}, $T(n)$ can be written as

$$T(n) = T_{\text{setup}} + \frac{T_{\text{compute}}(1)}{n} + T_{\text{finalize}} \tag{3}$$

The efficiency can be expressed as

$$E(P) = \frac{S(P)}{P} \tag{4}$$

and gives an idea of how good a parallel code works. If the efficiency is close to 1, the parallelization is very good. The theoretically possible value for $E = 1$ is called a *perfect linear speedup* [5].

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA019

Work Depth Model

The Work Depth Model, described by Blelloch [6], allows to compare the execution time of parallel algorithms. Especially when using graphs for parallelization, like the parameter hierarchy in LSA, other comparison mechanism do not fit to the problem. For example, Amdahl's Law assumes that the code is equally parallelized, but as the parameter hierarchy in LSA is an unbalanced Directed Acyclic Graph (DAG) this assumption is never fulfilled.

In the context of parallelizing LSA, the work W is expressed by the amount of settings and the depth D is expressed by the depth of the parameter hierarchy. Using Eq. (5) of Blelloch, a range for time T can be calculated for a given number of processing units P where time T depends on the hardware:

$$\frac{W}{P} \leq T < \frac{W}{P} + D \ , \tag{5}$$

with respect to Eq. (1) one can say that the speedup in the Work Depth Model is

$$\frac{WP}{W + PD} \leq S(P) < P \ . \tag{6}$$

TEST SCENARIOS

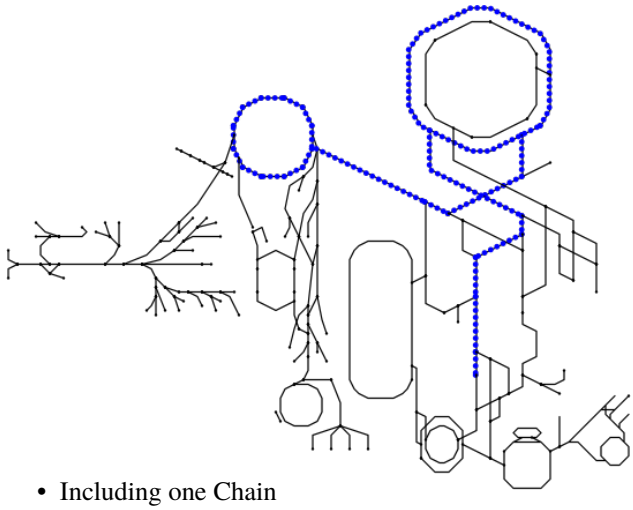
Since this thesis aims at optimizing the calculations performed in LSA, any other influence on the measurements must be ruled out. Therefore the patterns with all their Beam Production Chains (chains), beam processes and settings as well as the parameter hierarchies with all parameters are read out of the database before the calculations and time measurements were started. Also the database persistence was removed. This procedure is also called a "trim simulation" where only the LSA calculations are performed, but no data gets persisted in the database and no data is sent to the real accelerator devices.

The Figures 2 and 3 visualize the two patterns, creating four scenarios used for testing and Table 1 provides some more details about them.

Table 1: Overview of the Test Scenarios

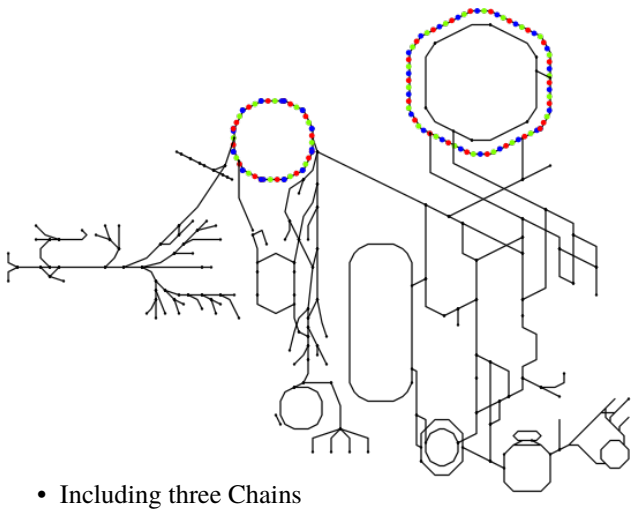
Scenario	Nr. of settings	No. of changed high level settings	No. of calculated dependent settings	average original time
P1-1	14 538	1	8 707	12.7 s
P2-1	38 943	1	11 184	132.9 s
P2-2	38 943	1	8 466	18.1 s
P2-3	38 943	2	19 650	155.2 s

Each scenario was run twice for warmup, to let the Just in Time (JIT) compiler of the Java Virtual Machine (JVM) optimize the code. On the first run, also the caches of LSA were filled, after the third run, the execution time for each trim was within an acceptable range caused by the normal fluctuations in execution time.



- Including one Chain
- Changed one high-level parameter in
  - SIS18 (P1-1)

Figure 2: Pattern 1.



- Including three Chains
- Changed one high-level parameter in
  - SIS100 (P2-1)
  - SIS18 (P2-2)
  - SIS18 and SIS100 (P2-3)

Figure 3: Pattern 2.

OPTIMIZATIONS

The following optimizations were performed

Sequential

- use caching where possible
- use suitable data structures for the main use case
- reduce array copies when inserting (or deleting) multiple points to a function
- change algorithms with complexity $O(n^2)$ to those with $O(n \log n)$ where possible
- do not calculate a setting for all its parents but only once all its parents have been calculated

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA019

Parallel

- block on cache request if another thread actually calculates the result
- run static data preparation in parallel
- run calculation loops in parallel where possible
- use parameter hierarchy as a task graph

OPTIMIZATION RESULTS

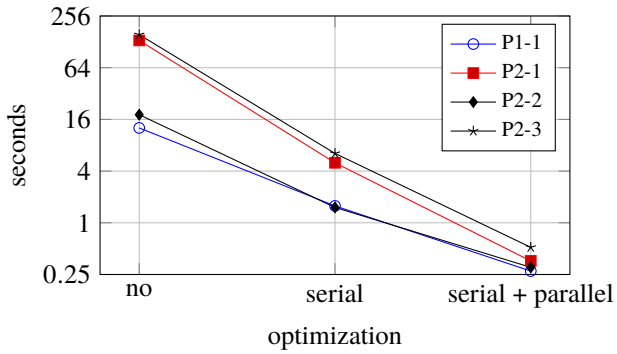


Figure 4: Average execution times on the target platform.

For Figure 4 each scenario was run twice for warmup and five times for measurements on the target platform (10 cores and Hyper Threading Technology (HT), 64 GB RAM). The parallel execution was measured with the default thread pool size of 19 plus the main thread. One can assume, that the patterns used for operating in the full FAIR accelerator complex will also experience an identical speedup.

Figures 5 and 7 show the memory consumption on the target platform for scenario P2-3. The scenario was run twice for warmup and five times for measurements.

The speedup for the test platform with 4 cores and HT is between 3.92 and 4.00 while the speedup range for the target platform are 9.49 to 10.00. The measured speedups

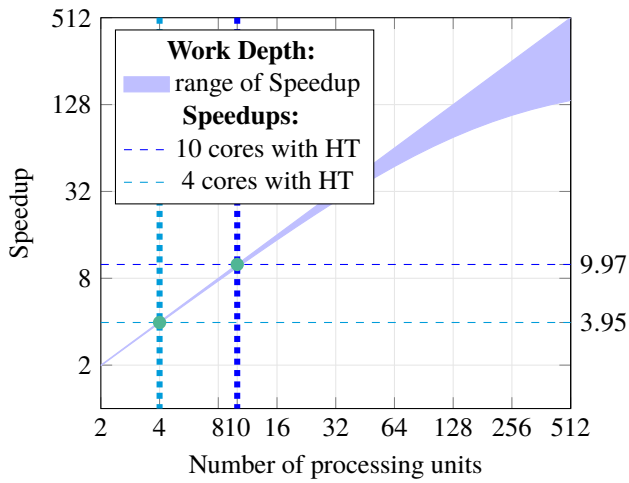


Figure 6: Work Depth Model: Equation (6) for $W = 3728$ (changed setting settings in SIS100) and $D = 20$ (depth of parameter hierarchy for SIS100).

3.95 on the test platform respectively 9.97 on the target platform are at the upper bound of the theoretical ranges. Since the trim calculates values for several thousand settings, one can assume that the theoretical forecast will also be seen in practice, if a platform with more cores is used.

The parallel speedup on the target platform with 10 cores and HT has an efficiency E of *0.997*, on the test platform with 4 cores and HT the efficiency is *0.987* (see Eq. (4)), which nearly is a so called *perfect linear speedup* where $E = 1$.

Figure 5 shows the unoptimized memory allocations. There were more that 5 GB memory allocated and immediately afterwards released and collected by the Garbage Collector (GC). For the five measured trims, the GC was executed 305 times deleting over 1.3 TB memory data. Whereas in Fig. 7 the memory usage was optimized and for each trim

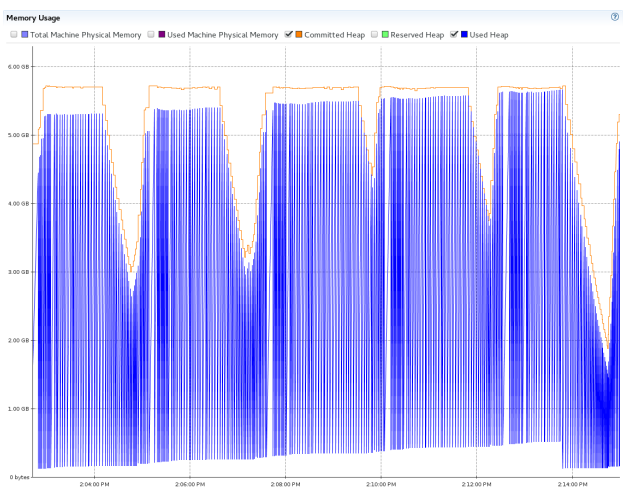


Figure 5: Memory usage on target platform without optimizations: 738.6 s, 305 garbage collections, 1 353.4 GB total memory allocation.

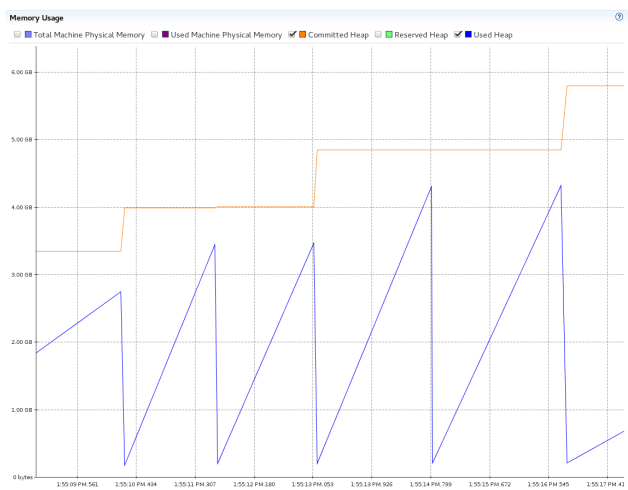


Figure 7: Memory usage on target platform with optimizations: 8.7 s, 5 garbage collections, 18.9 GB total memory allocation.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA019

there was only one GC execution, deleting in sum 19 GB of data.

For Fig. 8, each scenario was executed two times for warmup and five times for measurements on a test platform (4 cores, 12 GB RAM) with and without HT. The default thread pool size is $n - 1$, so for the setup without HT, the default thread pool size is 3 and with HT the default thread pool size is 7. One can clearly see, that the execution time reaches its limit with about eight threads, which is the double of the hardware threads. Afterwards the execution time stagnates or even gets worse. Also its indisputable that HT has a noticeable speedup for the calculations in LSA.

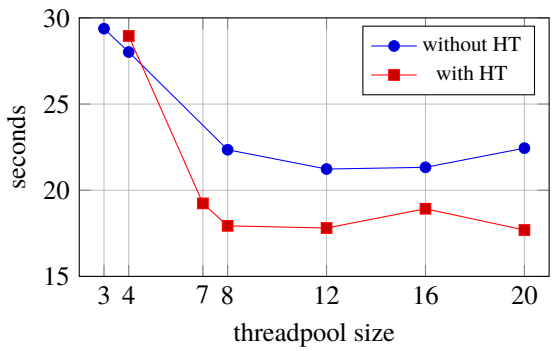


Figure 8: Execution times for different thread pool sizes.

Table 2 provides a detailed overview of the Central Processing Unit (CPU), GC, thread pool usage, Thread Local Allocation Buffer (TLAB) and allocation rates. The scenario P2-3 was therefore trimed two times for warmup and one time for measurement on the target platform. The parallel execution was done with the default JVM settings, so 19 threads in the thread pool plus the main thread. However, the number of available Java-threads can be much higher, depending on the tasks, so that the scheduler is able to use load-balancing. The TLAB is a storage each thread has for itself, read and write accesses to this storage do not need any synchronization. The TLAB size is automatically adjusted by the JVM; however there's the potential of optimization, if one knows how big the TLAB will be one could set this size as an argument to the JVM. The Object storage on the other hand is accessible by all threads and needs to be synchronized when multiple threads make a use of it. In the table one can clearly see, that the trim time reduces from 95.6s to 0.7s in the fully optimized version. Besides the memory usage, manifested in the TLAB size, which was also described in figures 5 and 7 the CPU consumption is another number to note. In the original version, only one of the ten cores, building two threads due to HT, was used where on the parallel version all cores were used and reached a workload of nearly 63 %. The thread pool now takes over 90 % of the total calculations.

Table 2: Overview of the Needed Resources for P2-3 on the Target Platform

Optimization	None	Sequential	Sequential & Parallel
trim time	95.6 s	5.4 s	0.7 s
CPU usage avg	6.2 %	9.1 %	62.8 %
CPU usage max	19.9 %	11.1 %	62.8 %
Heap avg	2.5 GB	1.8 GB	2.0 GB
Heap max	5.4 GB	3.4 GB	3.8 GB
GC pause time			
avg	11.4 ms	29.7 ms	33.8 ms
max	70.5 ms	29.7 ms	33.8 ms
Main thread usage	100 %	100 %	8.5 %
Thread pool usage			
total	0 %	0 %	91.5 %
avg	0 %	0 %	4.8 %
TLAB size			
total	147.5 GB	6.4 GB	4.0 GB
avg	64.1 MB	56.8 MB	3.6 MB
max	105.7 MB	75.0 MB	72.8 MB
Allocation rate for TLAB	1 495.0 MB/s	731.3 MB/s	672.0 MB/s
Object size			
total	114.3 kB	10.4 kB	5.0 kB
avg	1.0 kB	10.4 kB	0.8 kB
max	32.0 kB	10.4 kB	2.1 kB
Allocation rate for Objects	1.1 kB/s	1.2 kB/s	0.8 kB/s

SUMMARY

By reducing the memory consumption and complexity of the most used algorithms of LSA from $O(n^2)$ to $O(n \log n)$ the sequential calculation time could be sped up by 22.00. Parallelizing the DAG containing the parameter hierarchy and some loops in the trim calculations increased the speedup by a factor of 9.97 on the target platform with 10 cores with hyper threading. This leads to an average speedup of 219.23 which now allows the user to seamlessly change the overall accelerator scheduling.

In this paper, LSA was optimized at GSI. The next steps will be the merge with the common code base at European Organization for Nuclear Research (CERN). More about the collaboration of GSI and CERN can be found in [7].

REFERENCES

[1] G. Kruk *et al.*, "LHC Software Architecture (LSA) - Evolution Toward LHC Beam Commissioning", in *Proc. ICALEPCS'07*, Knoxville, TN, USA, paper WOPA03.

[2] GSI, "Cryring@ESR", https://www.gsi.de/en/work/research/appamml/atomic_physics/experimental_facilities/cryringesr.htm accessed on 2017-09-26

[3] GSI, "SIS18 Sections", https://www.gsi.de/en/work/accelerator/heavy_ion_synchrotron_sis18/sis18_sections.htm accessed on 2017-09-26

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA019

[4] FAIR, "Heavy Ion Synchrotron SIS100", http://www.fair-center.eu/news-events/news-view/article/heay-ion-synchrotron-sis100.html accessed on 2017-09-26

[5] G. T. Mattson, A. B. Sanders, and B. L. Massingill, *Patterns for Parallel Programming*, Addison-Wesley 2010, 6th Printing, ISBN 0321228111

[6] G. E. Blelloch, "Programming parallel algorithms", *Communications of the ACM*, 1996, http://portal.acm.org/citation.cfm?doid=227234.227246,

[7] J. Fitzek *et al.*, "First Production Use of the New Settings Management System for FAIR", presented at ICALEPCS'17, Barcelona, Spain, Oct 2017, paper THPHA062.