Design and Implementation of a Low-overhead Run-time System for Self-X Architectures

Diplomarbeitsvortrag

Mario Kicherer

Universität Karlsruhe (TH)
Institut für Technische Informatik
Lehrstuhl für Rechnerarchitektur
Prof. Dr. Wolfgang Karl

29. Januar 2009

Motivation

- growing amount of parallel heterogeneous and highly dynamic systems
- hardware configuration and the individual usage can change, especially during run-time

→ Applications have to be adjustable to changing circumstances for compatibility and maximum performance.

→ The implementation of a specific functionality has to be exchangeable to adapt to the respective environment.

Possible solution: Virtual Machines

- existing virtual machines introduce high amount of complexity

→ decreased efficiency for high-performance computation

→ not suitable for limited hardware, e.g. embedded systems

Motivation

- growing amount of parallel heterogeneous and highly dynamic systems
- hardware configuration and the individual usage can change, especially during run-time

→ Applications have to be adjustable to changing circumstances for compatibility and maximum performance.

→ The implementation of a specific functionality has to be exchangeable to adapt to the respective environment.

Possible solution: Virtual Machines

- existing virtual machines introduce high amount of complexity

→ decreased efficiency for high-performance computation

→ not suitable for limited hardware, e.g. embedded systems

Motivation

- growing amount of parallel heterogeneous and highly dynamic systems
- hardware configuration and the individual usage can change, especially during run-time

→ Applications have to be adjustable to changing circumstances for compatibility and maximum performance.

→ The implementation of a specific functionality has to be exchangeable to adapt to the respective environment.

Possible solution: Virtual Machines

- existing virtual machines introduce high amount of complexity

→ decreased efficiency for high-performance computation

→ not suitable for limited hardware, e.g. embedded systems

Principles

- avoid additional complexity (for low-overhead)
- allow the exchange of functionality during run-time (for Self-X capabilities)
- provide control over the exchange process to a special authority (for Self-X capabilities)

Basic mechanism

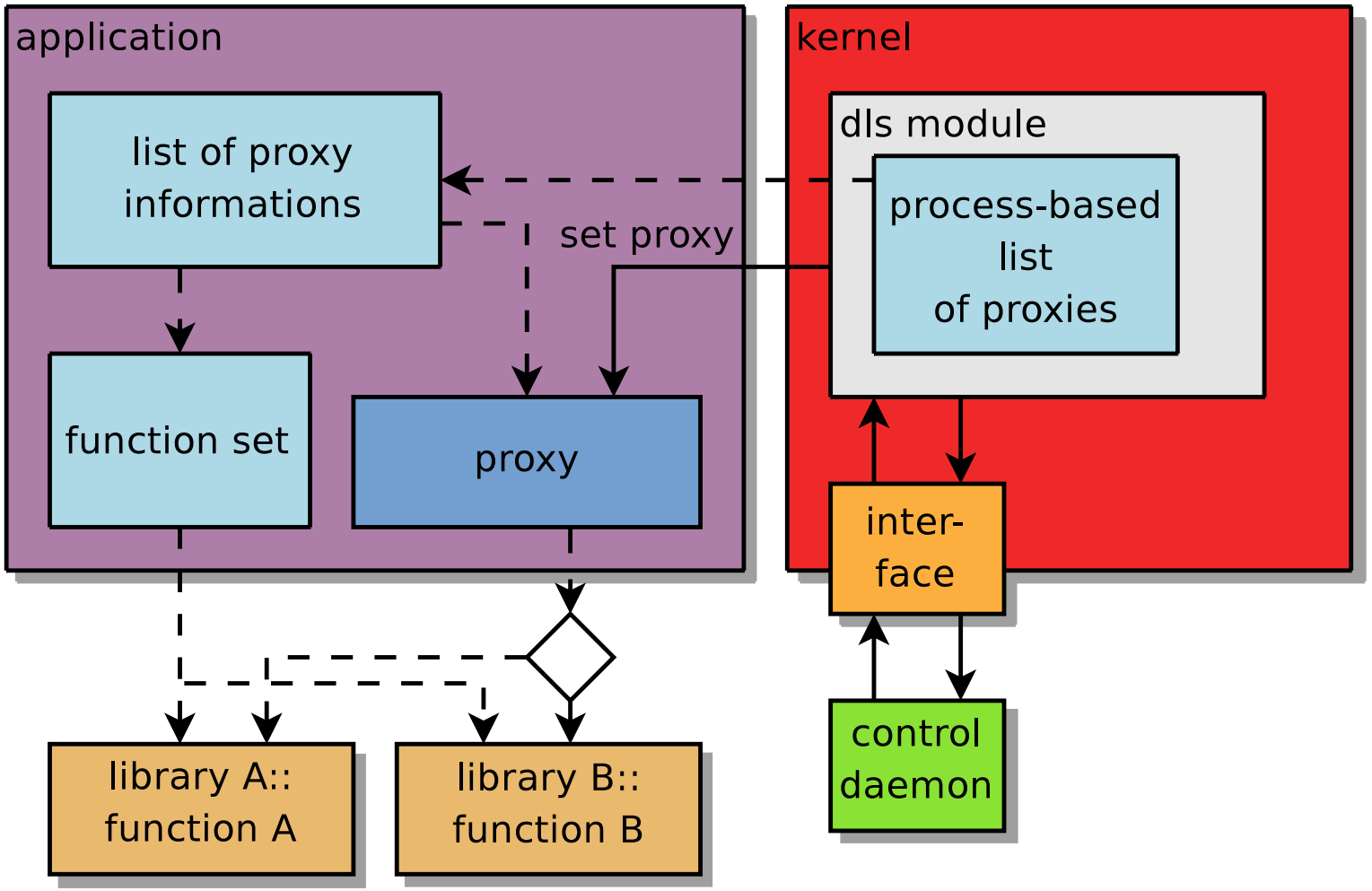
- on-demand rerouting of function calls

$\longrightarrow$ Ability to choose the appropriate implementation, encapsulated in a function

Dynamic Linking System (DLS)

- realizes the basic mechanism in the source code
- A function call is replaced by an indirect call through a function pointer, further called “proxy”.
- Preparation of the source code with the aid of three macros:
  - define the proxy and the corresponding management structures
  - initialize them
  - add a function alternative
- tasks of the kernel:
  - maintain a list of proxies on a per-process base
  - execute the switch algorithm
  - provide an interface to user space
- proxies controllable on a per-thread base

Schematic overview



DLS versions

Static loading (DLS-SL)

- the linker resolves the symbols

\+ fast

– Every library has to be available on the file system during linking and in the address space during execution.

Dynamic loading (DLS-DL)

- resolves the symbols during run-time using the dynamic linking loader

\+ Only the currently required libraries have to reside in the address space.

– Additional time overhead is created for the resolve process.

DLS versions

Static loading (DLS-SL)

- the linker resolves the symbols

\+ fast

– Every library has to be available on the file system during linking and in the address space during execution.

Dynamic loading (DLS-DL)

- resolves the symbols during run-time using the dynamic linking loader

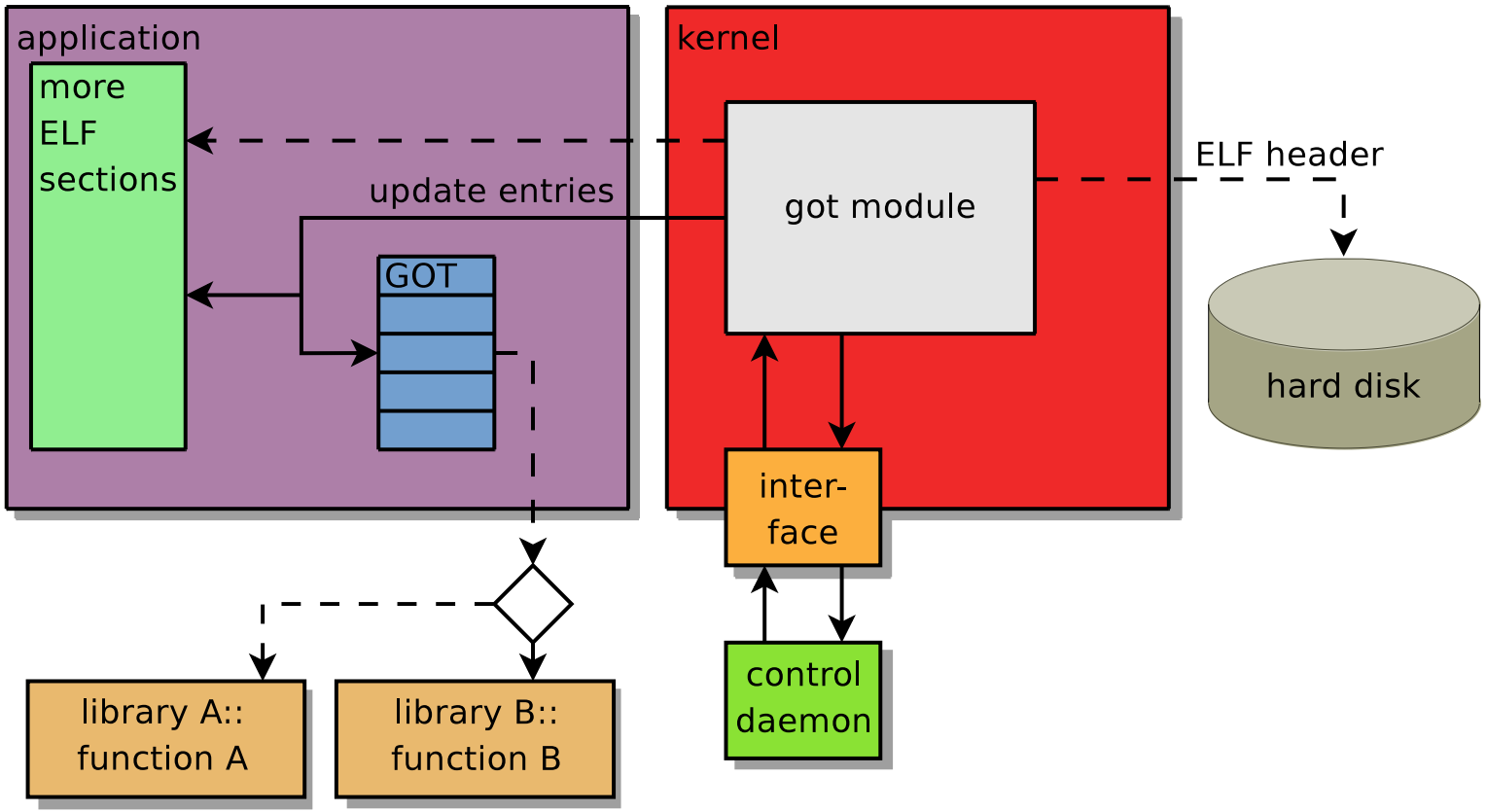
\+ Only the currently required libraries have to reside in the address space.

– Additional time overhead is created for the resolve process.

GOT-based Linking System (GLS)

- realizes the basic mechanism using ELF structures
- The Executable and Linkable Format (ELF) is the standard format for binaries on modern Linux systems.
- GOT is part of the ELF specification, stands for “Global Offset Table” and stores the addresses of symbols.
- The kernel contains the switch logic and provides an interface to user space.
- rerouting only controllable on a per-process base

Schematic overview



Discussion I

DLS

+ fast (DLS-SL)
+ low memory footprint (DLS-DL)
+ fine-grained control on a per-call-origin and per-thread base
+ possibility to trigger an in-application switch
– modifications in application source code necessary
– modifications in kernel source code necessary

Discussion II

GLS

+ compatible with proprietary applications
+ independent of compiler and source language
+ the whole system fits in a kernel module
− only coarse-grained control

Pure overhead

- sample application for stress testing
- function call in a `for` loop with 10.000.000 iterations
- two equal libraries, containing only the following function

```
double cpuMul(double a, double b) {
        return a * b;
}
```

	average	overhead
GLS	21.59s	-
DLS-SL	21.52s	$\sim$ 0%
DLS-DL	21.53s	$\sim$ 0%

Results II

Run-times with a switch after every call:

	average	overhead
DLS-SL (in-application)	26.48s	22%
DLS-DL (in-application)	54.72s	152%
DLS-SL (kernel)	37.20s	71%
GLS (kernel)	63.90s	195%
DLS-DL (kernel)	69.48s	220%

Results III

Calculated time overhead for a single switch:

In-application	Delay	Kernel	Delay
DLS-SL	0.5μs	DLS-SL	1.6μs
DLS-DL	3.3μs	DLS-DL	4.8μs
		GLS	4.1μs

Comparison

Initialization of a CUDA kernel: 0.6 seconds

Results III

Calculated time overhead for a single switch:

In-application	Delay	Kernel	Delay
DLS-SL	0.5μs	DLS-SL	1.6μs
DLS-DL	3.3μs	DLS-DL	4.8μs
		GLS	4.1μs

Comparison

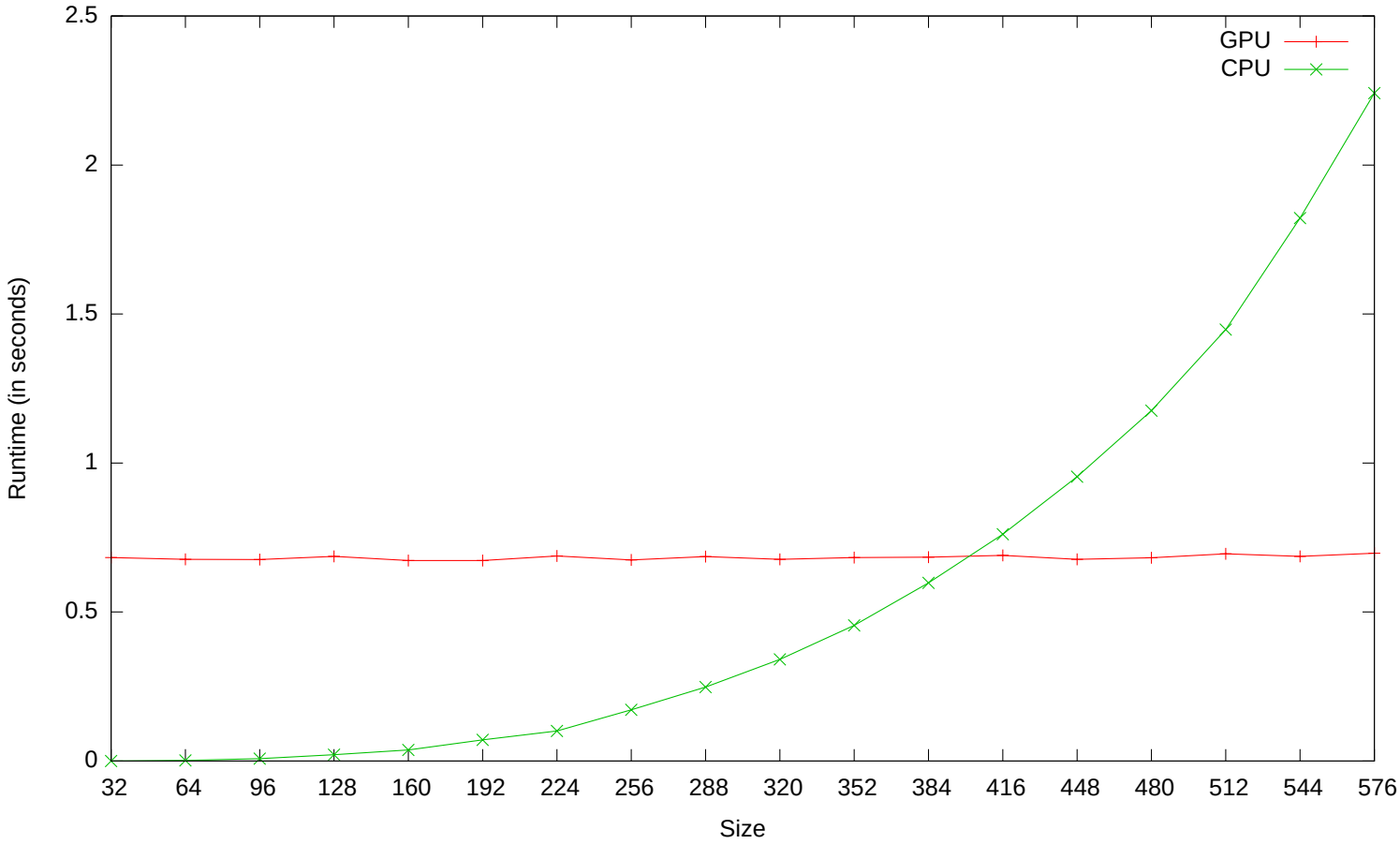
Initialization of a CUDA kernel: 0.6 seconds

Parallel heterogeneous application

- application with real world background from the numerical mathematics area
- calculates a square matrix multiplication on either a multi-core CPU using OpenMP or a Nvidia GPU using CUDA

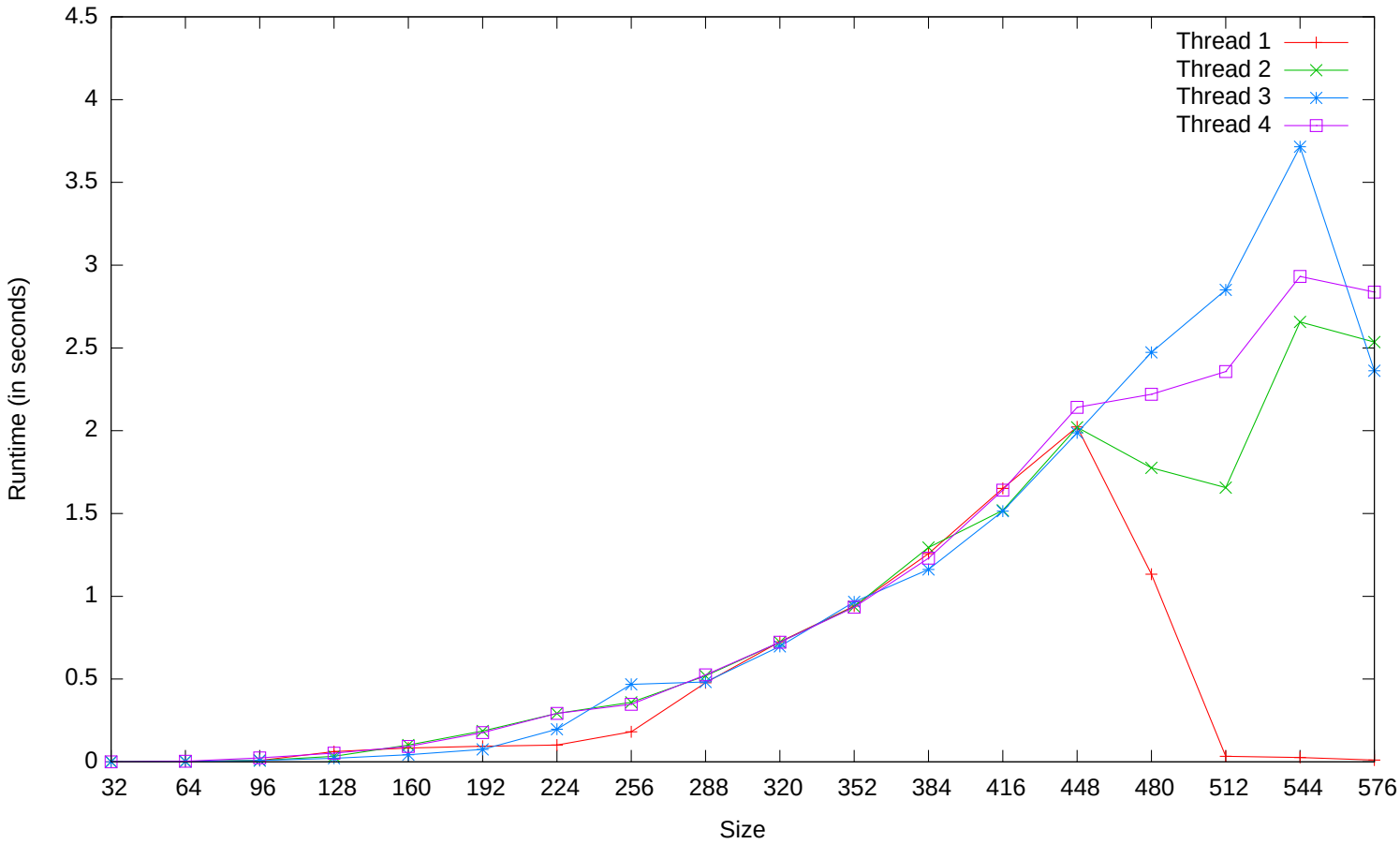
Results I

Time consumption of single square matrix multiplications:



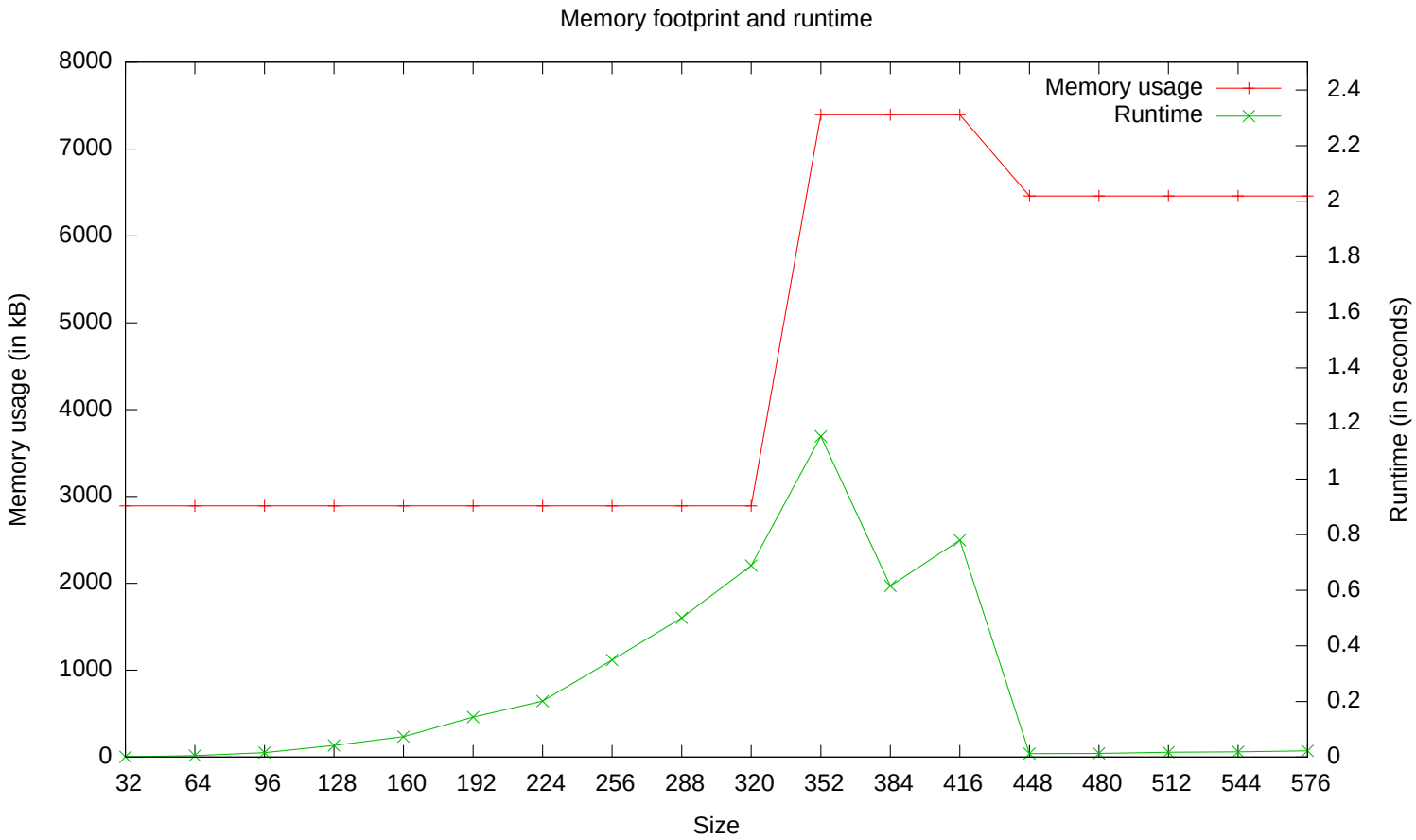
Results II

Time consumption of 4 parallel threads calculating square matrix multiplications with increasing size and one CPU → GPU switch:



Results III

Time and memory consumption of a similar application that calculates two multiplications in a row:



Conclusion

- uses regular system mechanisms
- small additional complexity

$\longrightarrow$ no measurable overhead through preparation

$\longrightarrow$ minor time overhead through switch mechanism

- ability to exchange of functionality during run-time
- controllable through a kernel interface, usable by a user space daemon

Outlook

Near

- Adding function alternatives during run-time
- Advanced control daemon

Far

- Distribution of computation on other nodes

Outlook

Near

- Adding function alternatives during run-time
- Advanced control daemon

Far

- Distribution of computation on other nodes

Design and Implementation of a Low-overhead Run-time System for Self-X Architectures

Diplomarbeitsvortrag

Mario Kicherer

Universität Karlsruhe (TH)
Institut für Technische Informatik
Lehrstuhl für Rechnerarchitektur
Prof. Dr. Wolfgang Karl

29. Januar 2009