Technische Universität Berlin

Scalarization and Temporal SIMT in GPUs: Reducing Redundant Operations for Better Performance and Higher Energy Efficiency

Jan Lucas | TU Berlin - AES

LPGPU

AES Embedded Systems Architecture

Technische Universität Berlin

Overview

- What is a Scalarization?
- Why are Scalar Operations are common in GPU Kernels?
- Scalarization and Register Allocation
- Scalarization in Conventional GPUs
- Scalarization in GPUs using Temporal SIMT
- Energy and Performance Results

LPGPU AES Embedded Systems Architecture

Technische Universität Berlin

What is a Scalarization?

- GPUs execute bundles of threads called Warps on SIMD units
- Operation is always the same for all threads and sometimes data matches as well
- This causes redundant operations and storage
- If we can guarantee that the input data of an instruction is always the same, we can eliminate this redundancy and save energy and storage space
- We do not to add want to detect this at runtime, but move all the hard work to the compiler

LPGPU AES Embedded Systems Architecture

Why are Scalar Operations are common in GPU Kernels?

- Recalculating values often cheaper than sharing them between different threads
- Lots of small calculations do not depend on threadIdx in any way

Example:

```
int qryid = __umul24(blockIdx.x, blockDim.x) + threadIdx.x;
```

- Does not look that Scalar?

Why are Scalar Operations are common in GPU Kernels?

Example:

```
int qryid = __umul24(blockIdx.x, blockDim.x) + threadIdx.x;
```

Does not look scalar but actually contains many scalar instructions!

```
cvt.u32.u16 %r1, %ctaid.x;
cvt.u32.u16 %r2, %ntid.x;
mul24.lo.u32 %r3, %r1, %r2;
cvt.u32.u16 %r4, %tid.x;
add.u32 %r5, %r4, %r3;
```

How to identify Scalar Instructions?

~~How~~ Where to identify Scalar Instructions?

- We do not want to do a complicated anaylsis at runtime.
- We can only reduce the number of registers needed per warp, if we can identify Scalar values before we launch the kernel.

- We should analyze Scalarization using as static algorithm.
- Should be done by the CPU: compiler or driver
- Immediate forms such as PTX or SPIR provide enough information

- Here: Implemented as a preprocessing step at the kernel load on PTX in gpgpu-sim.

Technische Universität Berlin

How can we identify Scalar instructions?

First step (dataflow):

1. Optimistically mark all registers as scalar
2. Mark all destination registers of instruction using threadidx or non-scalar values as vector
3. Repeat Step 2. until no new vector registers are found.

This already gets most registers right, but unfortunately there is a second reason, why register values could indirectly depend on threadidx: Control Flow

LPGPU AES Embedded Systems Architecture

How can we identify Scalar instructions?

How can we deal with Control Flow and Scalarization?

- Lee et al.: Only scalarize if control flow is convergent.
  Correct & simple but suboptimal!

- We can do better, if we consider variable lifetime!
- Control flow does not have to be convergent as long as variable is dead in all other flows.

Lee et al. „Convergence and Scalarization for Data-Parallel Architectures“, CGO 2013

Control flow and Register Lifetime

```
for (int i=0;i<threadIdx.x;i++)
{
        ...
}
```

- Control flow is different in the different threads of the warp → not convergent
- BUT: i is dead after finishing the loop and i is the same for all threads that have not yet reached the reconvergence point at the end of the loop
- **We can scalarize i!** (Lee et al. cannot!)

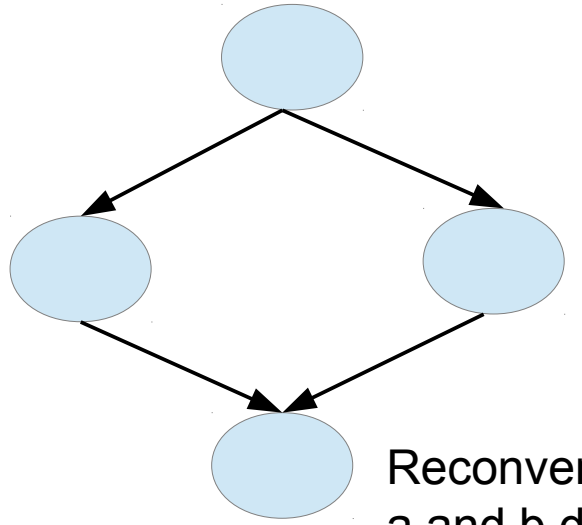
Scalarization and Register Allocation

- Register lifetime is critical for Scalarization!
- Use a static single assignment(SSA) or similar representations for analyzing Scalarness.
- When we allocate registers, the register allocation needs to be aware of Scalarization
- Not just register liveness within a single thread is important but liveness within the warp also needs to be considered.

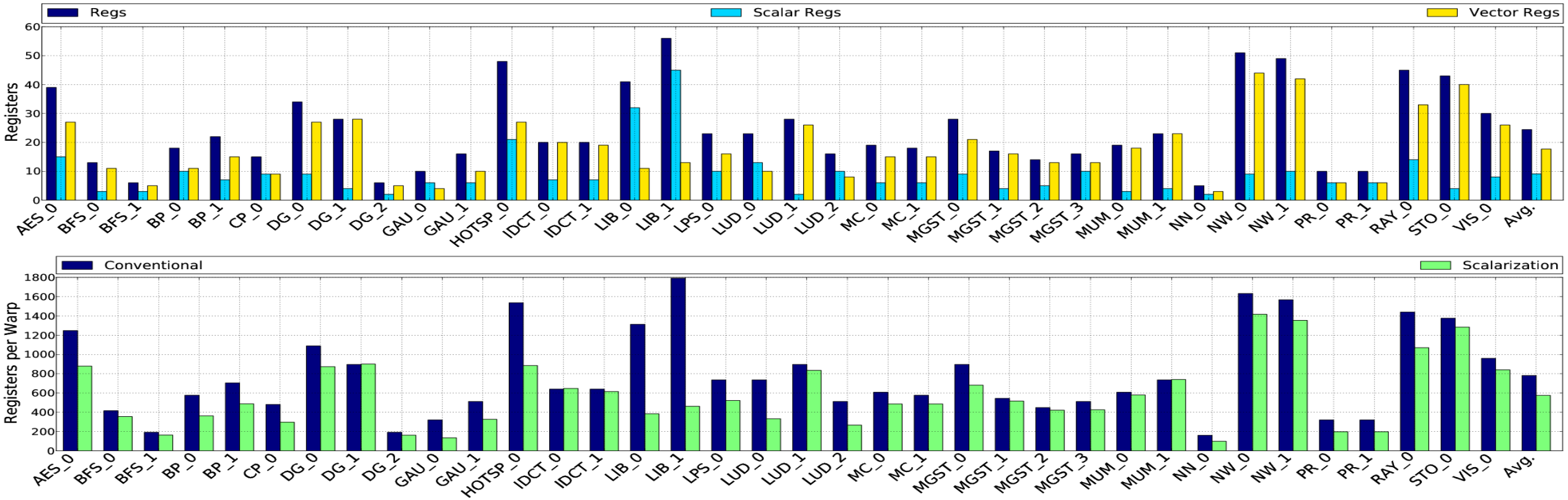
Scalarization and Register Allocation

```
if (threadIdx.x %2)
{
        (...) = a;
} else
{
        b=(scalar value);
}
```

- a and b are never alive at the same time within a single thread, but we cannot assign the same register to them

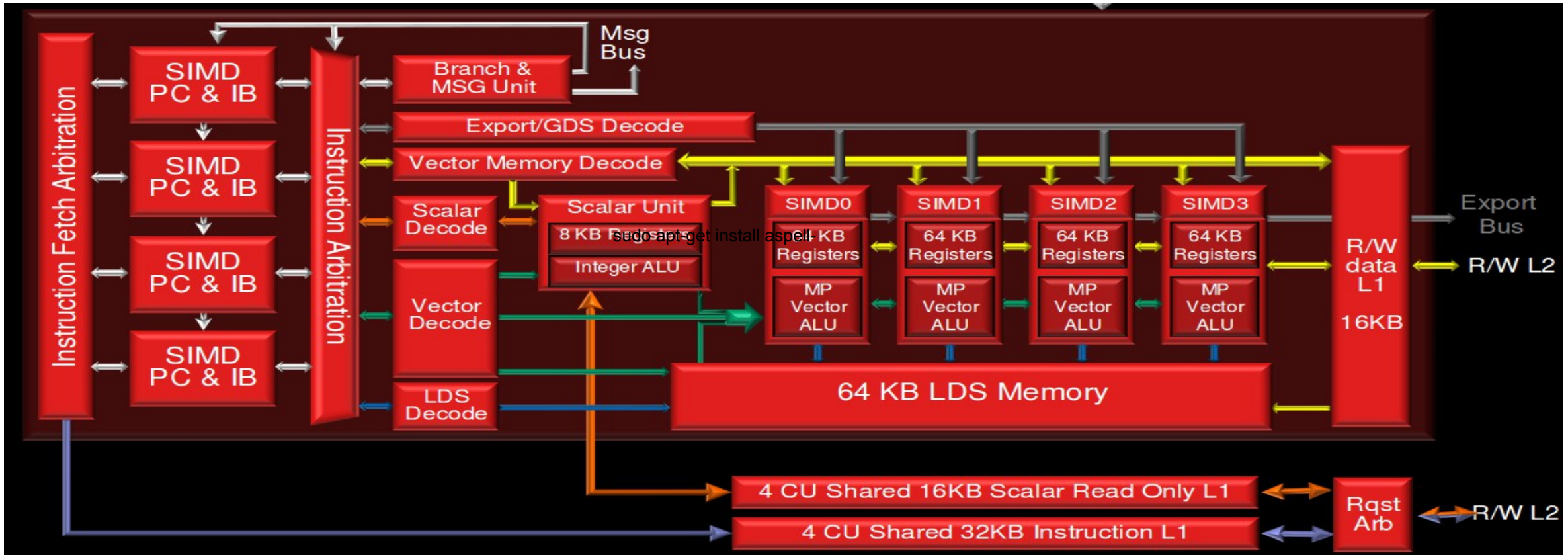


Results of the Static Analysis



- 26.5% fewer registers required per warp

Scalarization in AMDs GCN Architecture



sudo apt-get install aspell

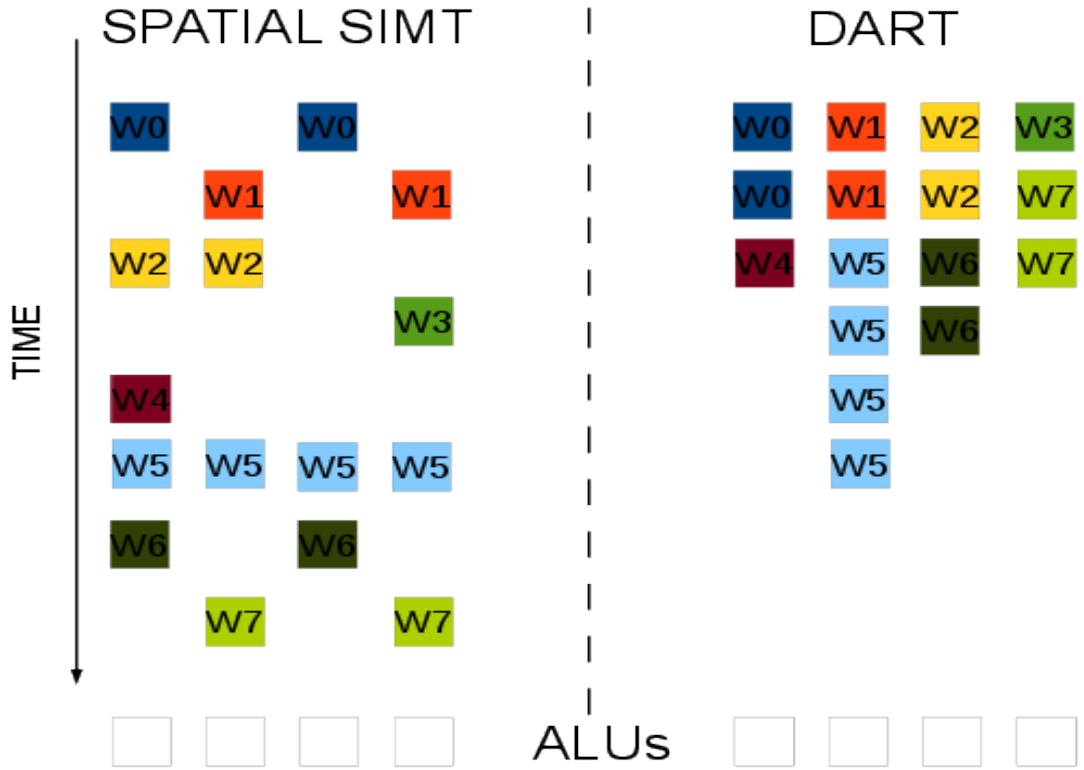
Source: AMD

Issues with Scalarization in Conventional GPUs

- Scalarization in Conventional GPUs requires many additional hardware structures
  - Separate Register Files(RFs) and Execution Units
  - Broadcast networks to distribute scalar results to SIMD Units
- Execution Units and RFs can only execute/store Scalar or Vector Instructions/Values
  - Idle Execution Units, if Scalar/Vector Unit Ratio does not match code
  - Either: Unused Registers in Scalar-RF
  - Or: Not all Scalar Values fit in Scalar-RF, cannot fully exploit the potential of Scalarization

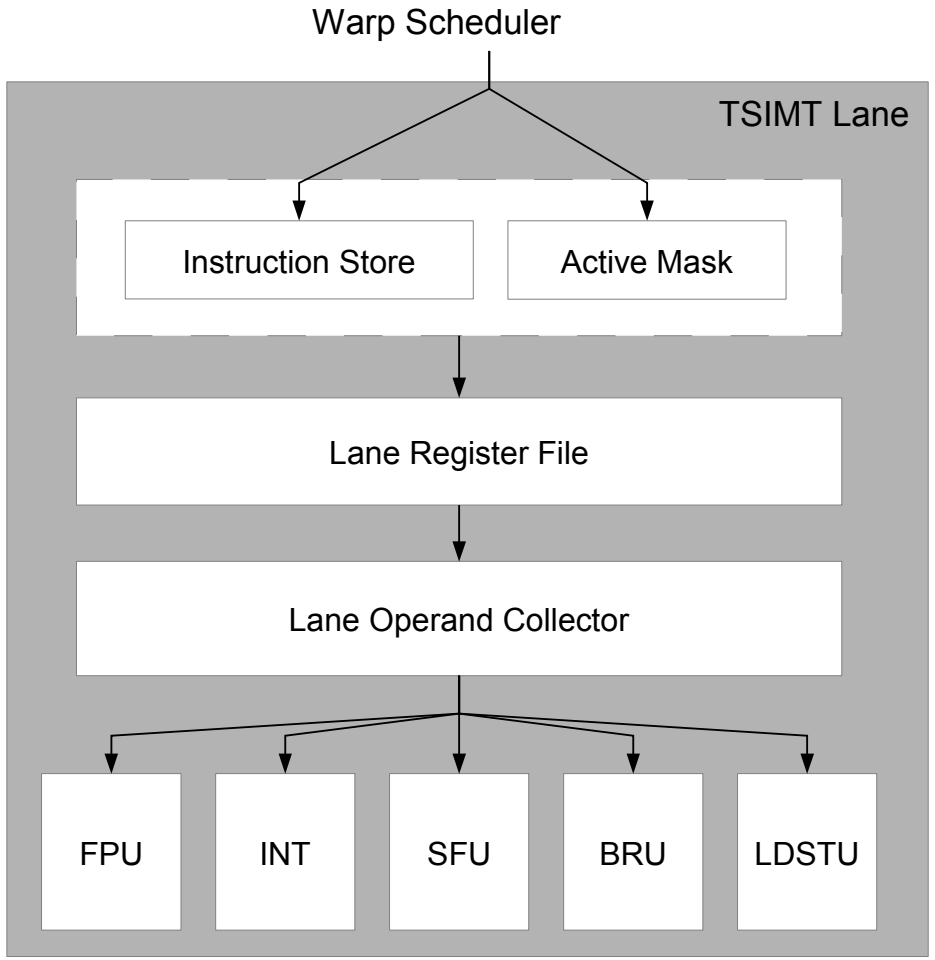
Temporal SIMT

- Instead of execution of SIMT code on spatial SIMD units,
  execute on a single execution unit over multiple cycles
- Similar to 1-lane vector processors
- Unused Slots can be skipped
- Use multiple parallel lanes with a shared frontend to archive high performance



Temporal SIMT & Scalarization

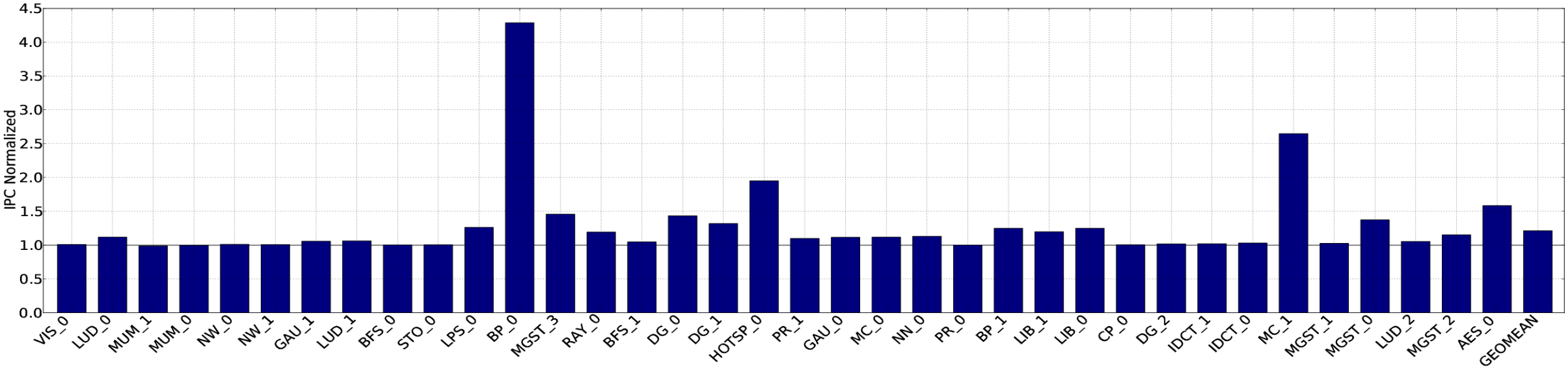
- Temporal SIMT makes hardware support for Scalarization easy!
- No broadcast network needed!
- Scalar and Vector values can be stored in the same register file
- Scalar Operations are executed on the same ALU as Vector Operations, almost like a warp with only one active thread
- Tiny LUT (64-128bit) with one bit per register can be used to control scalarness



Experimental Setup

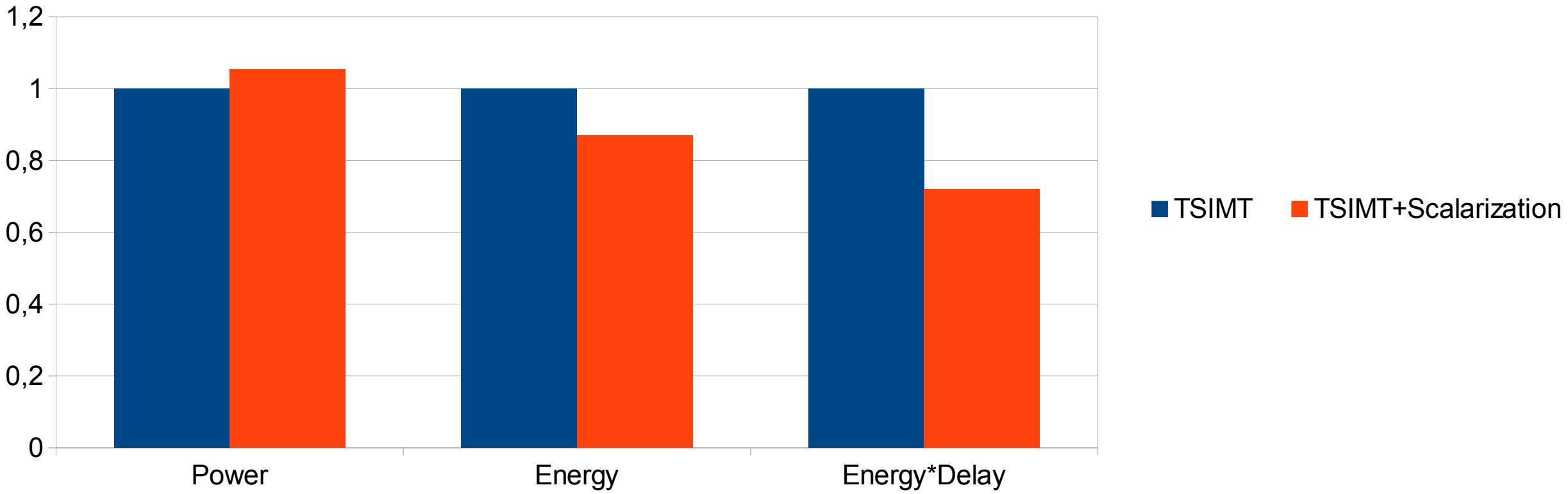
- Benchmarks from Rodina, CUDA Toolkit, LPGPU
- gpgpusim 3.2.1 based simulator
- We added support for the TSIMT-based DART architecture to this simulator
- Scalarization Support added to simulator
- Extended gpusimpow power simulator
- GTX580 based configuration:15 Cores, 8 Lane per Core

Results – Performance



- Average Speedup: 1.22X
- 12 Kernels with small or no benefits (~1.0X speedup)
- 17 Kernels with significant benefits (1.1-1.5X speedup)
- 4 Kernels with huge benefits (>1.5-4.2X speedup)

Results - Energy



Technische Universität Berlin

Conclusions

- Scalarization+Register Allocation should be done together
- Scalarization is easy to implement in TSIMT-based GPUs
- Scalarization has significant power and performance benefits

LPGPU AES Embedded Systems Architecture