Performance Portability Analysis for Real-Time Simulations of Smoke Propagation using OpenACC

Anne Küsters*, **Sandra Wienke+**, Lukas Arnold*

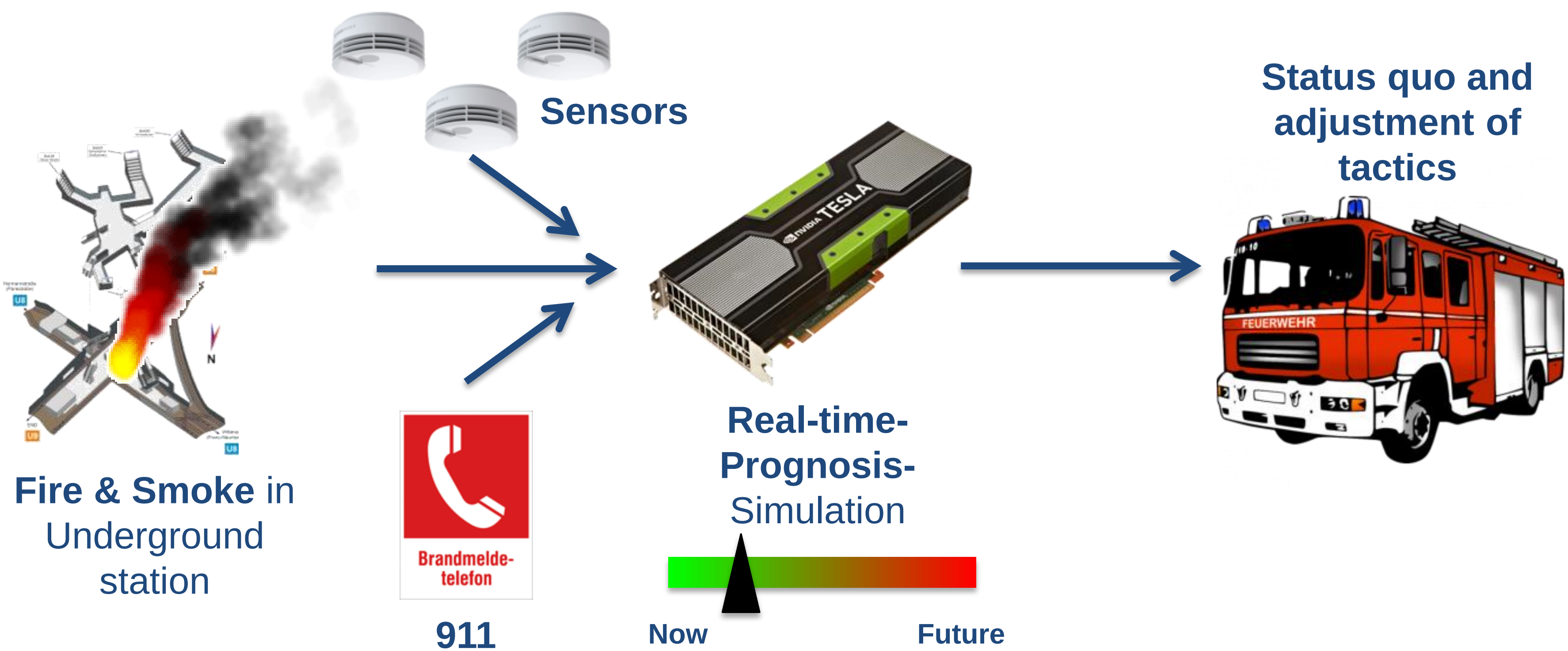
*Forschungszentrum Jülich
+RWTH Aachen University

June 22nd, 2017

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

Motivation

- **Real-time simulation tool to support fire fighters**
- ***JuROr*: Prediction of smoke propagation in complex rooms**
  - Parallelization with OpenACC



Motivation

- **Today: variety of hardware architectures**
- **Can (PGI’s) OpenACC provide performance portable code?**

- **Analysis of JuROr’s performance portability: Roofline Model**
  - Model: manually-computed arithmetic intensity
  - Real-world code (!): measured arithmetic intensity

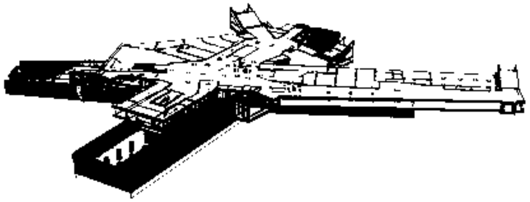
- **Investigation of various hardware architectures**
  - Three NVIDIA GPUs, four Intel CPUs

Agenda

- **Real-time Simulation of Smoke Propagation**
- **Parallelization with OpenACC**
- **Analysis of Performance Portability with Roofline Model**
- **Results**
- **Conclusion and Outlook**

JuROr – Simulation of Smoke Propagation

- **CFD Solver**
  - Navier-Stokes equations with weakly compressible smoke formulation
  - Simplification by fractional step method
  - Finite Differences (structured grid)
    - Advection: Semi-Lagrangian Method
    - Diffusion: implicit Jacobi-Method
    - Sources: explicit Euler (BD) in t
    - Pressure: Incompressibility, Multigrid, Jacobi method & Projection
  - Turbulence with LES Model



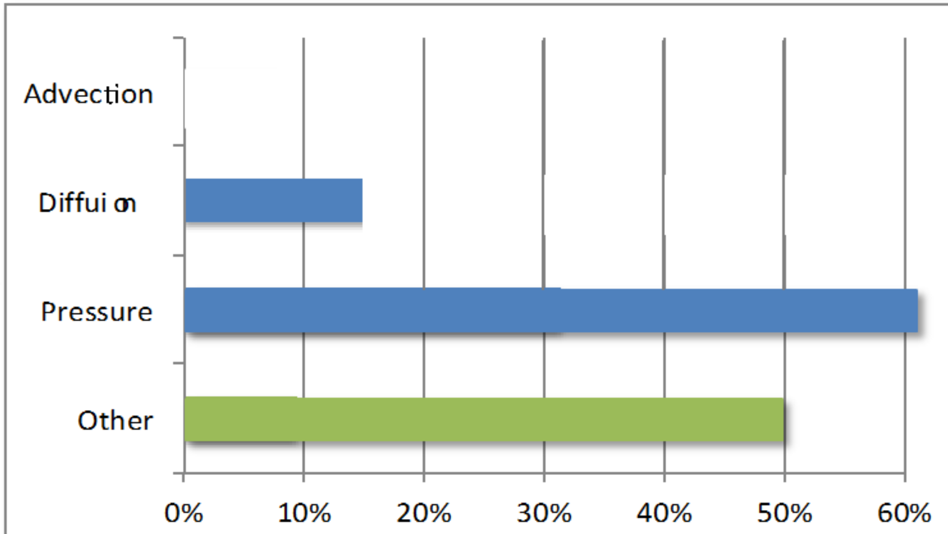
- **Test Case (for benchmarking)**
  - 2D Navier-Stokes equation (DP) in a $[0; 2\pi]^2$ square
  - Uniform grid (max $4096 \times 4096 \approx 135$ MB)

JuROr – Simulation of Smoke Propagation

Pseudocode

```
while t<t_end do
      advect()

      diffuse()

      calcPressureAndProject()

      other()

end
```

Performance shares



- **Major driver: solving Laplacian equation (Jacobi stencil)**
  – Roofline Model focuses on Jacobi Step

Offloading Kernels (keeping max freedom)

CPU (C++)

```
void Jacobi(double* out, double* in, double* b)
{
// local variables
// highly parallel for-loop




  for (int j = 0; j < Ny; j++){


    for (int i = 0; i < Nx; i++){
       out[i,j]= beta * (b[i,j]
            + alphaX * (in[i+1,j] + in[i-1,j]) \
            + alphaY * (in[i,j+1] + in[i,j-1]);
    }
  }


}
```

OpenACC

```
void Jacobi(double* out, double* in, double* b)
{
// local variables
// highly parallel for-loop
#pragma acc data copy(out[:size],in[:size],b[:size])
 {
#pragma acc kernels
#pragma acc loop independent
  for (int j = 0; j < Ny; j++){
#pragma acc loop independent
    for (int i = 0; i < Nx; i++){
       out[i,j]= beta * (b[i,j]
            + alphaX * (in[i+1,j] + in[i-1,j]) \
            + alphaY * (in[i,j+1] + in[i,j-1]);
    }
  }

 } // pragma acc data
}
```

Reduction of Launch Latency (async clause)

CPU (C++)

```
void Jacobi(double* out, double* in, double* b)
{
// local variables
// highly parallel for-loop




  for (int j = 0; j < Ny; j++){

    for (int i = 0; i < Nx; i++){
       out[i,j]= beta * (b[i,j]
            + alphaX * (in[i+1,j] + in[i-1,j]) \
            + alphaY * (in[i,j+1] + in[i,j-1]);
    }
  }


}
```

OpenACC

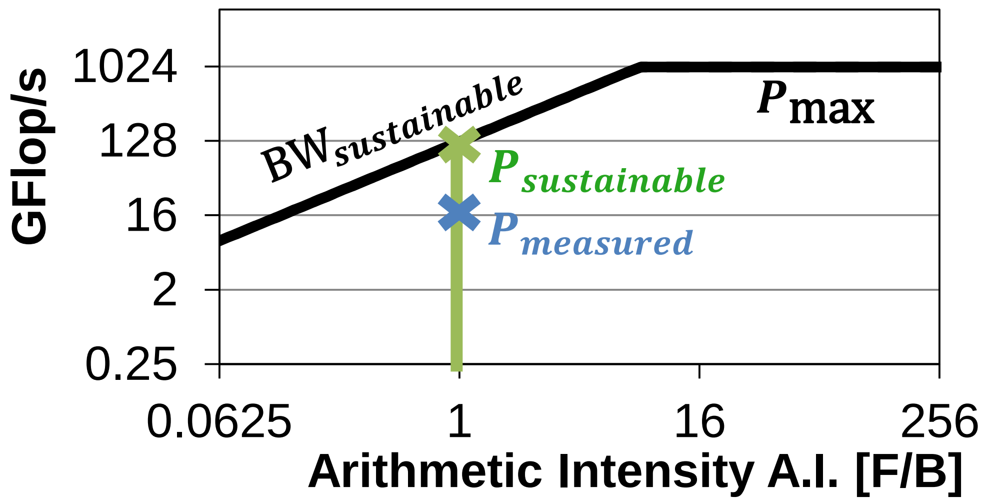
```
void Jacobi(double* out, double* in, double* b)
{
// local variables
// highly parallel for-loop
#pragma acc data copy(out[:size],in[:size],b[:size])
 {
#pragma acc kernels async
#pragma acc loop independent
  for (int j = 0; j < Ny; j++){
#pragma acc loop independent
    for (int i = 0; i < Nx; i++){
       out[i,j]= beta * (b[i,j]
            + alphaX * (in[i+1,j] + in[i-1,j]) \
            + alphaY * (in[i,j+1] + in[i,j-1]);
    }
  }
#pragma acc wait
 } // pragma acc data
}
```

RWTH AACHEN UNIVERSITY

JÜLICH FORSCHUNGSZENTRUM

ORPHEUS

Roofline Model



- **"Theoretical A.I."**
  - Manually count Flops and Bytes in code
  - Feasible for small kernels only
- **"Measured A.I."**
  - Measure Flops and Bytes (perf. counters)
  - Feasible for real-world codes

$$P_{sustainable}\,[\text{GFlop/s}] = \min(P_{\max},\ \text{A.I.} \cdot \text{BW}_{sustainable})$$

A.I.: [F/B]

$\text{BW}_{sustainable}$: [GB/s]

$$P_{share}\,[\%] = \frac{P_{measured}}{P_{sustainable}}$$

Hardware: Setup

Hardware	Used	Compiler	Flags `-fast -O3`
2-socket Intel Xeon **Broadwell (BDW)** E5-2650 v4 @2.20 GHz, 2x12 cores	1 socket	PGI 16.10	`-ta=multicore`
2-socket Intel Xeon **Haswell (HSW)** E5-2680 v3 @2.50 GHz, 2x12 cores	1 socket	PGI 16.1	`-ta=multicore`
2-socket Intel Xeon **Sandy Bridge (SNB)** E5-2650 0 @2.00 GHz, 2x8 cores	1 socket	PGI 16.1	`-ta=multicore`
2-socket Intel Xeon **Ivy Bridge (IVB)** E5-2640 v2 @2.00 GHz, 2x8 cores	1 socket	PGI 16.1	`-ta=multicore`
NVIDIA Pascal **P100** SMX2 GPU, 1328 MHz, 16 GB, autoboost N/A, ECC on, Broadwell host	1 GPU	PGI 16.10	`-ta=tesla:cc60`
NVIDIA Kepler **K80** with 2 GPUs, 562 MHz, 2x12 GB, autoboost off, ECC on, Haswell host	1 GPU	PGI 16.1	`-ta=tesla:cc35`
NVIDIA Kepler **K40** GPU, 745 MHz, 12 GB, autoboost N/A, ECC on, Sandy Bridge host	1 GPU	PGI 16.1	`-ta=tesla:cc35`

Hardware: Performance Limiters

Machine	Peak GFlop/s	$BW_{sustainable}$ [GB/s]
BDW	422.40	68.00
HSW	240.00	61.00
SNB	128.00	43.00
IVB	128.00	43.00
P100	4759.55	550.35
K80	935.17	149.70
K40	1430.40	191.20

- **Sustainable bandwidth by**
  - GPU-STREAM
  - Intel micro-benchmarks (slightly higher values than regular STREAM)
- **Variety in peak performances & bandwidth measures**
- **Remark: either CPU or GPU (no hybrid investigations)**
  - Abstraction of performance model including data transfers

Arithmetic Intensity (code's hotspot)

Theoretical A.I.

$$\text{A.I.} = \frac{\text{hotspot's floating-point operations}}{\text{hotspot's data movement}} = \frac{12\ \text{Flops}}{(1\ \text{read} + 1\ \text{write}) \cdot 8\ \text{Bytes}}$$

$$= 0.75\,\frac{\text{F}}{\text{B}}$$

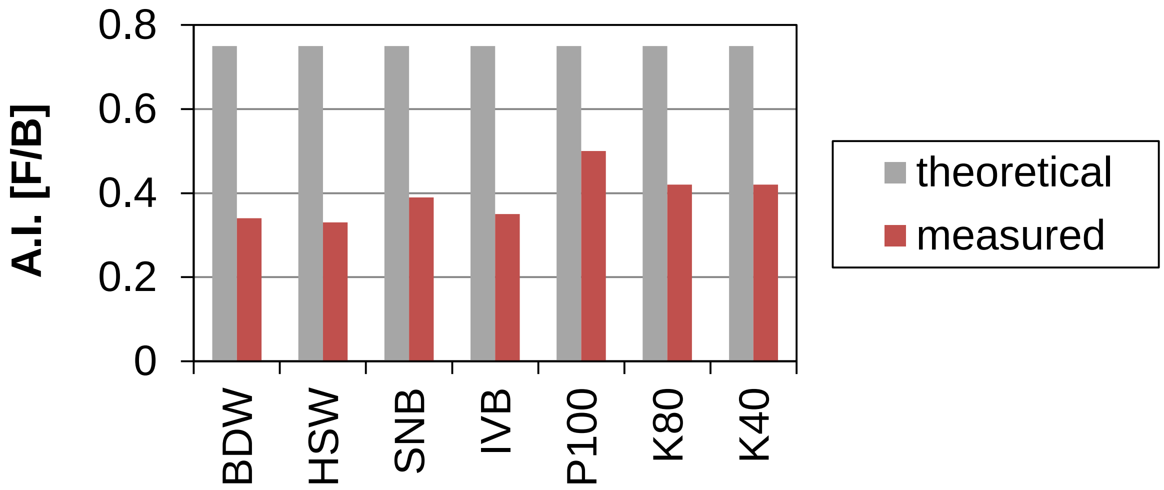
Measured A.I.

$$\text{A.I.}_{CPU} = \frac{\text{SCALAR} + \text{SSE\_PACKED} \cdot 2 + \text{256\_PACKED} \cdot 4}{(\text{read} + \text{write}) \cdot 64\ \text{Bytes}}$$

$$= \frac{\text{SCALAR} + \text{SSE\_PACKED} \cdot 2 + \text{256\_PACKED} \cdot 4}{\text{BW} \cdot \text{runtime}_{hotspot}}$$

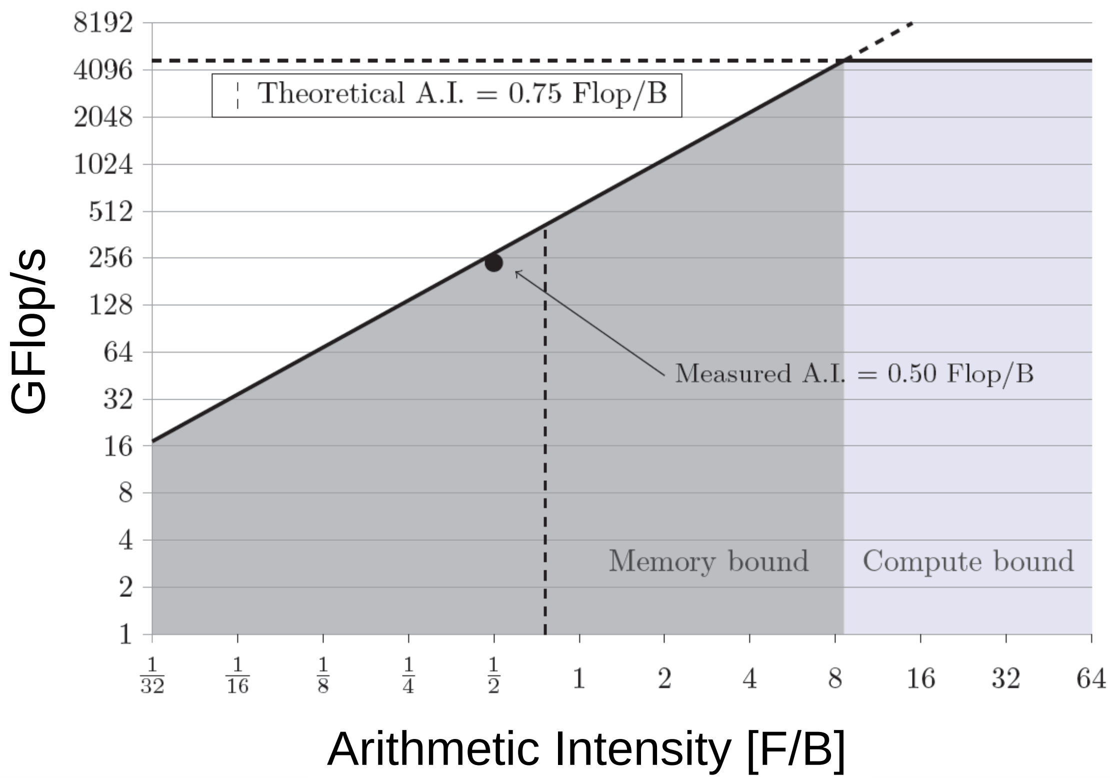
$$\text{A.I.}_{GPU} = \frac{\text{flop\_count\_dp}}{(\text{read} + \text{write}) \cdot 32\ [\text{threads per warp}]}$$

Arithmetic Intensity: Theoretical vs. Measured



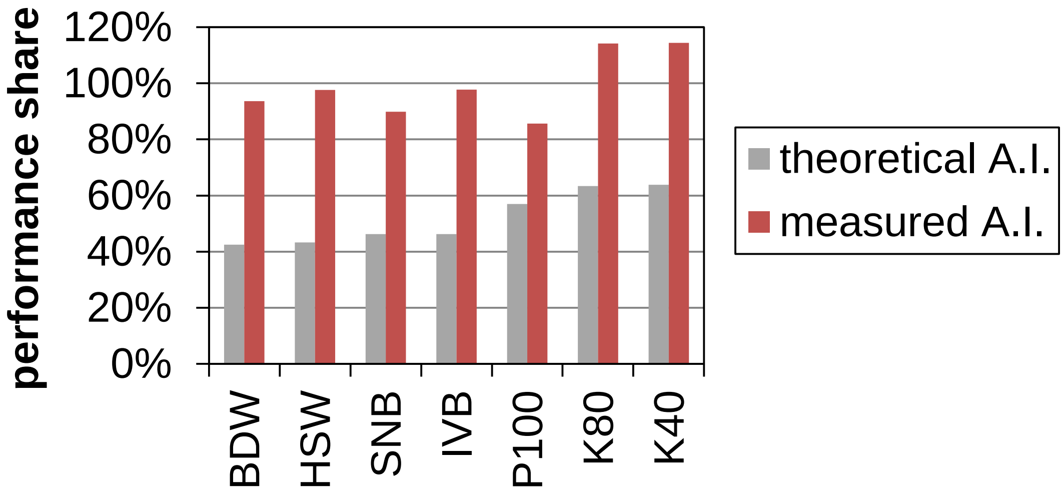
- **Measured A.I. roughly half of theoretical A.I.**
- **Measured A.I. only little deviation across all architectures**

Roofline Model (exemplary): NVIDIA P100



- **Performance limiter on all architectures: bandwidth**

Performance Shares



- **Similar performance shares across architectures**
  - Range: 86% - 114%
- **Good performance portability in one-source code**
  - Advection, diffusion, pressure and boundary condition in parallel
  - Possibility to maintain one source code base

Conclusion and Outlook

- **JuROr:** Prediction of **smoke propagation** based on **CFD**
- **Good performance portability** with our OpenACC parallelization (using PGI compiler)
  - Similar performance shares across seven architectures (Intel CPUs, NVIDIA GPUs)
  - Possibility to maintain one source codes base for these
- **Measured A.I. is reasonable** to use instead of theoretical A.I.
  - If code does not contain macho-Flop/s

- **Model data transfer** for roofline model
- Investigate OpenACC **performance on AMD GPUs**
- Continue to **develop 3D code** to handle 3D geometries
  - Geometries with obstacles and dynamic domain extension

Sources

[1] Doering, Gibbon, *Applied Analysis of the Navier-Stokes Equations*, Cambridge Texts in Applied Mathematics, (1995).

[2] S.L. Glimberg, K. Erleben, J. Bennetsen, *Smoke Simulation for Fire Engineering using a Multigrid Method on Graphics Hardware*, VRIPHYS, pp 11-20. Eurographics Association, (2009).

[3] Smagorinsky, Joseph, *General Circulation Experiments with the Primitive Equations*. Monthly Weather Review 91 (3): 99-164, (1991).