Performance of an Astrophysical Radiation Hydrodynamics Code under Scalable Vector Extension Optimization

Dennis Smolarski

Santa Clara University

Doug Swesty and Alan Calder

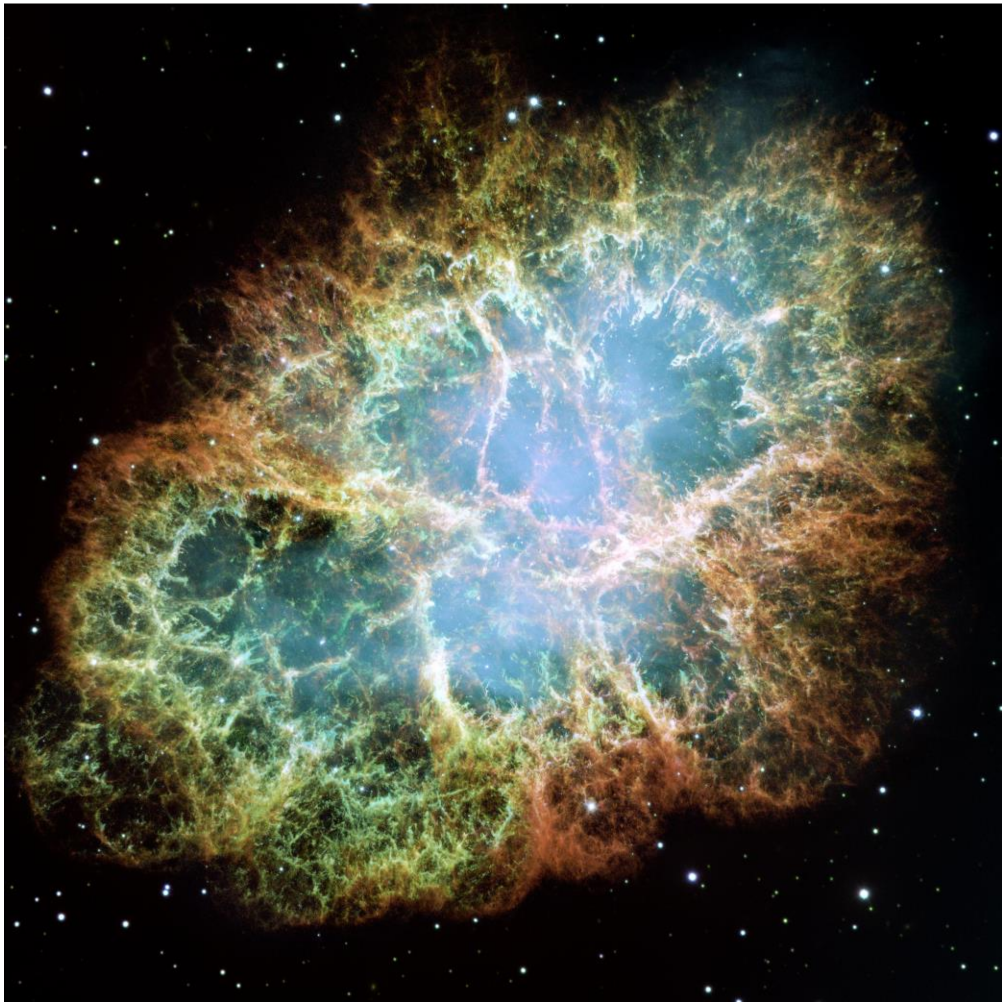
Stony Brook University

EAHPC-2022, 6 September 2022

Outline

- Core Collapse Supernovae
- Ookami--our A64-FX machine
- Radiation Transfer and V2D
- Tests on Ookami
- Conclusions

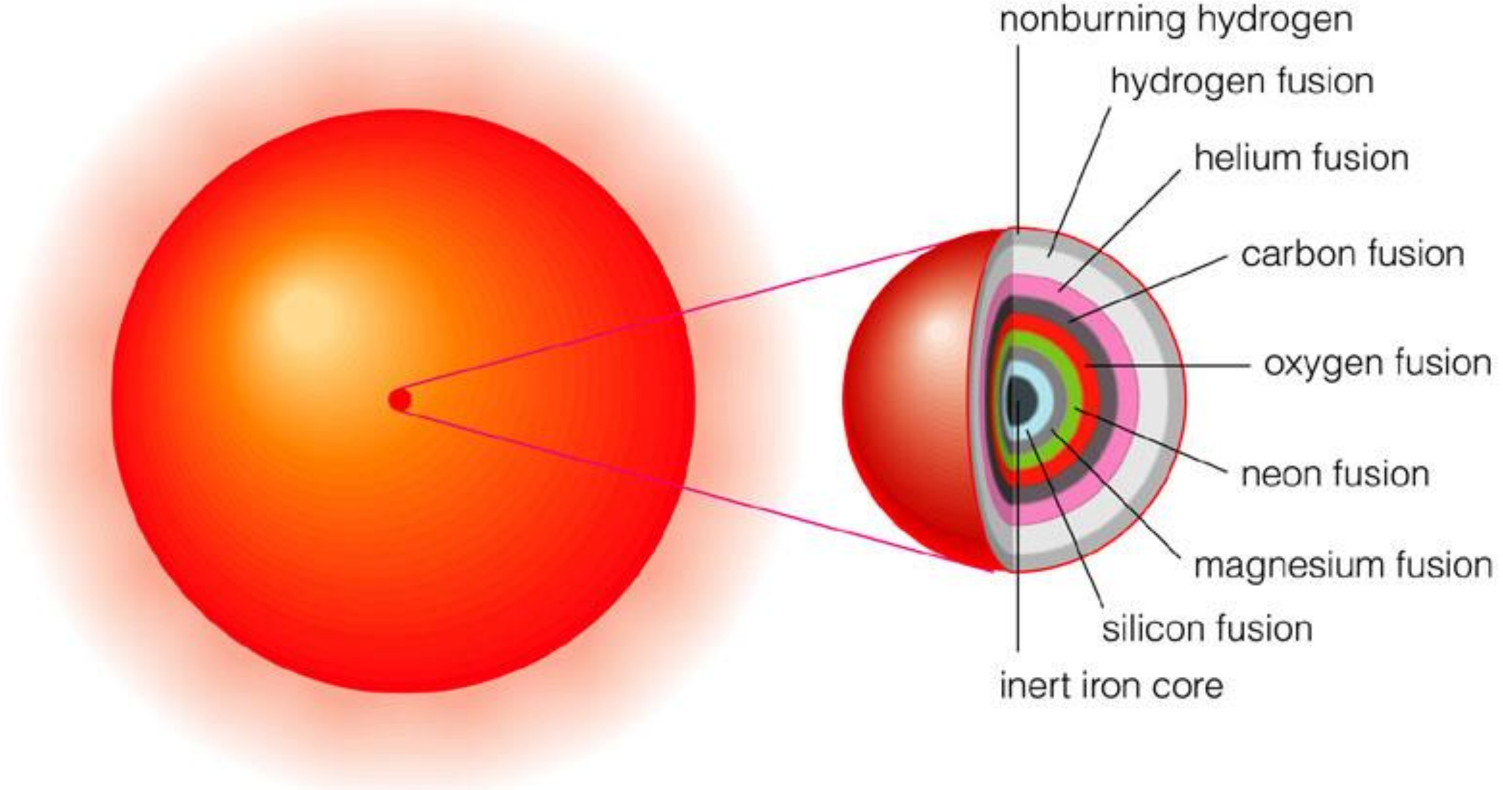
Core Collapse Supernovae



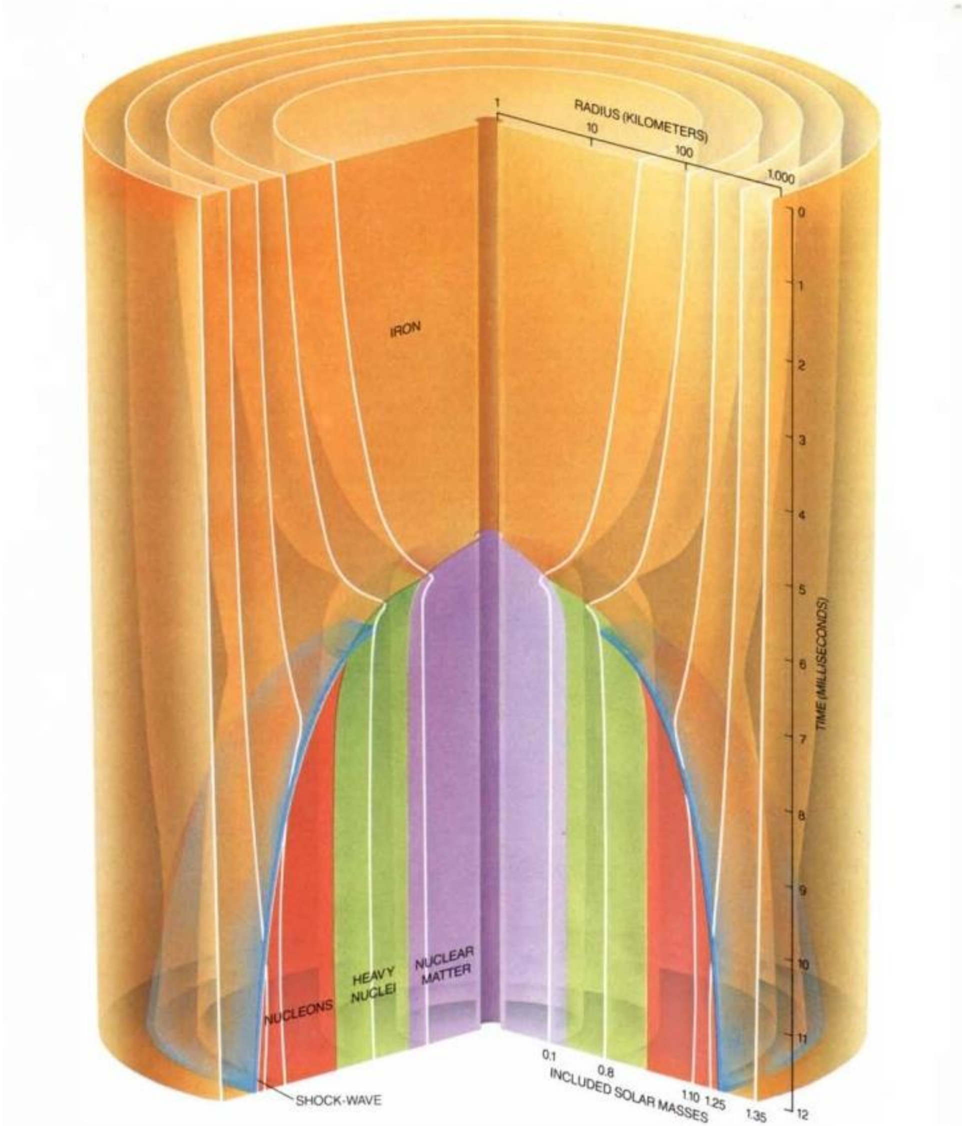
- Violent death of a massive (> 10 M   ) star.
- Powered by gravity
- Involve almost all branches of physics
- Marginal events
- Create neutron stars and black holes
- Neutrino radiation key

https://compstar.uni-frankfurt.de/outreach/short-articles/core-collapse-supernovae-fascinating-cosmic-fireworks/

Onion -like structure



World Diagram



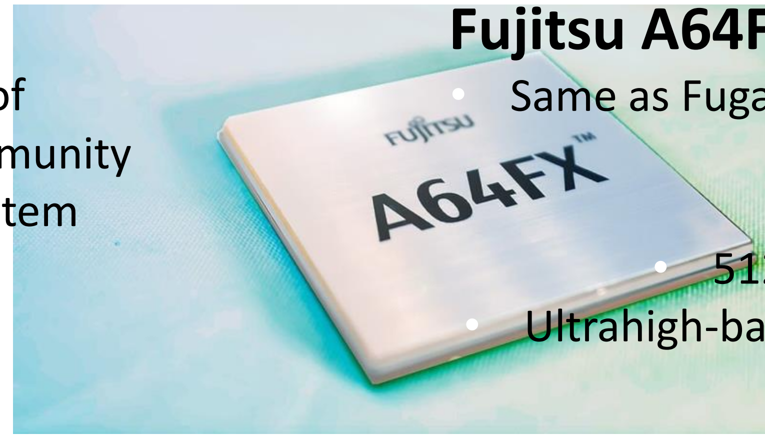
Bethe & Brown 1985

OOKAMI

A NEW PATH TO EXASCALE COMPUTING

Experimental testbed

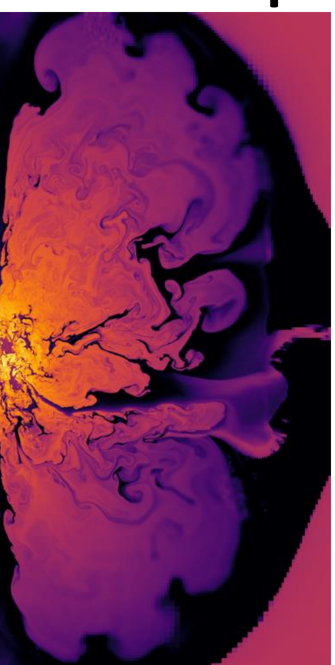
- First machine of its kind outside of Japan freely available to the community
- HPE (formerly Cray) Apollo 80 system
- Fall 2022: NSF XSEDE site



Fujitsu A64FX processors

- Same as Fugaku, #1 on Top500
- ARM-based
- Multi-core
- 512-bit SIMD-vector
- Ultrahigh-bandwidth memor

Current projects include:



- Type la supernovae
- Deep learning & AI
- Molecular dynamics
- Cancer cell simulations
- WarpX – E&M particle-in-cell
- Climate and weather modeling
- Plasma wakefield simulations
- Oceanic mesoscale mixing
- Black hole accretion
- Thermoelectric materials

Radiation Transfer and V2D

Work done by Doug Swesty and Eric Myra

$$\frac{1}{v}\frac{\partial I_\varepsilon}{\partial t} + \mu\frac{\partial I_\varepsilon}{\partial x} + I_\varepsilon(\mu)2\pi\int_{-1}^{1}\kappa^s(\mu,\mu')d\mu' - 2\pi\int_{-1}^{1}\kappa^s(\mu',\mu)I_\varepsilon(\mu')d\mu' + I_\varepsilon(\mu)\kappa^a(\mu) = S_\varepsilon(t,x,\mu)$$

$I_\varepsilon = I(t,\varepsilon,x,\mu)$	is the monochromatic intensity of the radiation
ε	is the energy of radiation "particles" at point x
$\cos\mu$	is the direction cosine
$\kappa^a(\mu)$ and $\kappa_s(\mu,\mu')$	are the absorption and scattering opacities

Radiation Transfer in V2D

$$\frac{\partial E_\varepsilon}{\partial t} + \boldsymbol{\nabla} \cdot (E_\varepsilon \mathbf{v}) + \boldsymbol{\nabla} \cdot \mathbf{F}_\varepsilon - \varepsilon \frac{\partial}{\partial \varepsilon} \left(\mathsf{P}_\varepsilon : \boldsymbol{\nabla} \mathbf{v}\right) = \mathbb{S}_\varepsilon,$$

$$\frac{\partial \bar{E}_\varepsilon}{\partial t} + \boldsymbol{\nabla} \cdot (\bar{E}_\varepsilon \mathbf{v}) + \boldsymbol{\nabla} \cdot \bar{\mathbf{F}}_\varepsilon - \varepsilon \frac{\partial}{\partial \varepsilon} \left(\bar{\mathsf{P}}_\varepsilon : \boldsymbol{\nabla} \mathbf{v}\right) = \bar{\mathbb{S}}_\varepsilon.$$

E_ε and $\bar{E}_\varepsilon$ are the particle and antiparticle monochromatic radiation-energy densities at point x

$\mathbf{F}_\varepsilon$ and $\bar{\mathbf{F}}_\varepsilon$ are the particle and antiparticle monochromatic flux densities at point x

$\mathbf{F}_\varepsilon$ and $\bar{\mathbf{F}}_\varepsilon$ are the particle and antiparticle radiation pressure tensors

$\mathbb{S}_\varepsilon$ and $\bar{\mathbb{S}}_\varepsilon$ are the particle and antiparticle radiation pressure tensors

Radiation Transfer in V2D

$$\frac{[E_\varepsilon]^{n+1}_{k+(1/2),i+(1/2),j+(1/2)} - [E_\varepsilon]^{n}_{k+(1/2),i+(1/2),j+(1/2)}}{\Delta t} - [\boldsymbol{\nabla} \cdot D_\varepsilon \nabla E_\varepsilon]^{n+1}_{k+(1/2),i+(1/2),j+(1/2)}$$
$$- \left[\varepsilon \frac{\partial\left(\mathsf{P}_\varepsilon : \boldsymbol{\nabla}\mathbf{v}\right)}{\partial\varepsilon}\right]^{n+1}_{k+(1/2),i+(1/2),j+(1/2)} - [\mathbb{S}_\varepsilon]^{n+1}_{k+(1/2),i+(1/2),j+)1/2)} = 0$$

This represents the linearization of the previous equations. Thus we can represent the system as *Ax*-*b* = 0, and use linear algebraic solvers to find the solution.

Radiation Transfer in V2D

$$
\begin{aligned}
&[\nabla \cdot D_\varepsilon \nabla E_\varepsilon]^{n+1}_{k+(1/2),i+(1/2),j+(1/2)} \equiv \\
&\frac{1}{[g_2]_{i+(1/2)}\,[g_{31}]_{i+(1/2)}\,[g_{32}]_{j+(1/2)}} \Bigg\{ \frac{1}{[x_1]_{i+(3/2)} - [x_1]_{i+(1/2)}} \\
&\quad \times \Bigg([g_2]_{i+1}\,[g_{31}]_{i+1}\,[g_{32}]_{j+(1/2)}\,[D_\varepsilon(x_1)]^{n+t}_{k+(1/2),i+1,j+(1/2)} \\
&\quad \times \frac{[E_\varepsilon]^{n+1}_{k+(1/2),i+(3/2),j+(1/2)} - [E_\varepsilon]^{n+1}_{k+(1/2),i+(1/2),j+(1/2)}}{[x_1]_{i+(3/2)} - [x_1]_{i+(1/2)}} \\
&\quad - [g_2]_i\,[g_{31}]_i\,[g_{32}]_{j+(1/2)}\,[D_\varepsilon(x_1)]^{n+t}_{k+(1/2),i,j+(1/2)} \\
&\quad \times \frac{[E_\varepsilon]^{n+1}_{k+(1/2),i+(1/2),j+(1/2)} - [E_\varepsilon]^{n+1}_{k+(1/2),i-(1/2),j+(1/2)}}{[x_1]_{i+(1/2)} - [x_1]_{i-(1/2)}} \Bigg) \\
&\quad + \frac{1}{[x_2]_{j+(3/2)} - [x_2]_{j+(1/2)}} \\
&\quad \times \Bigg(\frac{[g_{31}]_{i+(1/2)}\,[g_{32}]_{j+1}}{[g_2]_{i+(1/2)}}\,[D_\varepsilon(x_2)]^{n+t}_{k+(1/2),i+(1/2),j+1} \\
&\quad \times \frac{[E_\varepsilon]^{n+1}_{k+(1/2),i+(1/2),j+(3/2)} - [E_\varepsilon]^{n+1}_{k+(1/2),i+(1/2),j+(1/2)}}{[x_2]_{j+(3/2)} - [x_2]_{j+(1/2)}} \\
&\quad - \frac{[g_{31}]_{i+(1/2)}\,[g_{32}]_{j}}{[g_2]_{i+(1/2)}}\,[D_\varepsilon(x_2)]^{n+t}_{k+(1/2),i+(1/2),j} \\
&\quad \times \frac{[E_\varepsilon]^{n+1}_{k+(1/2),i+(1/2),j+(1/2)} - [E_\varepsilon]^{n+1}_{k+(1/2),i+(1/2),j-(1/2)}}{[x_2]_{j+(1/2)} - [x_2]_{j-(1/2)}} \Bigg) \Bigg\},
\end{aligned}
$$

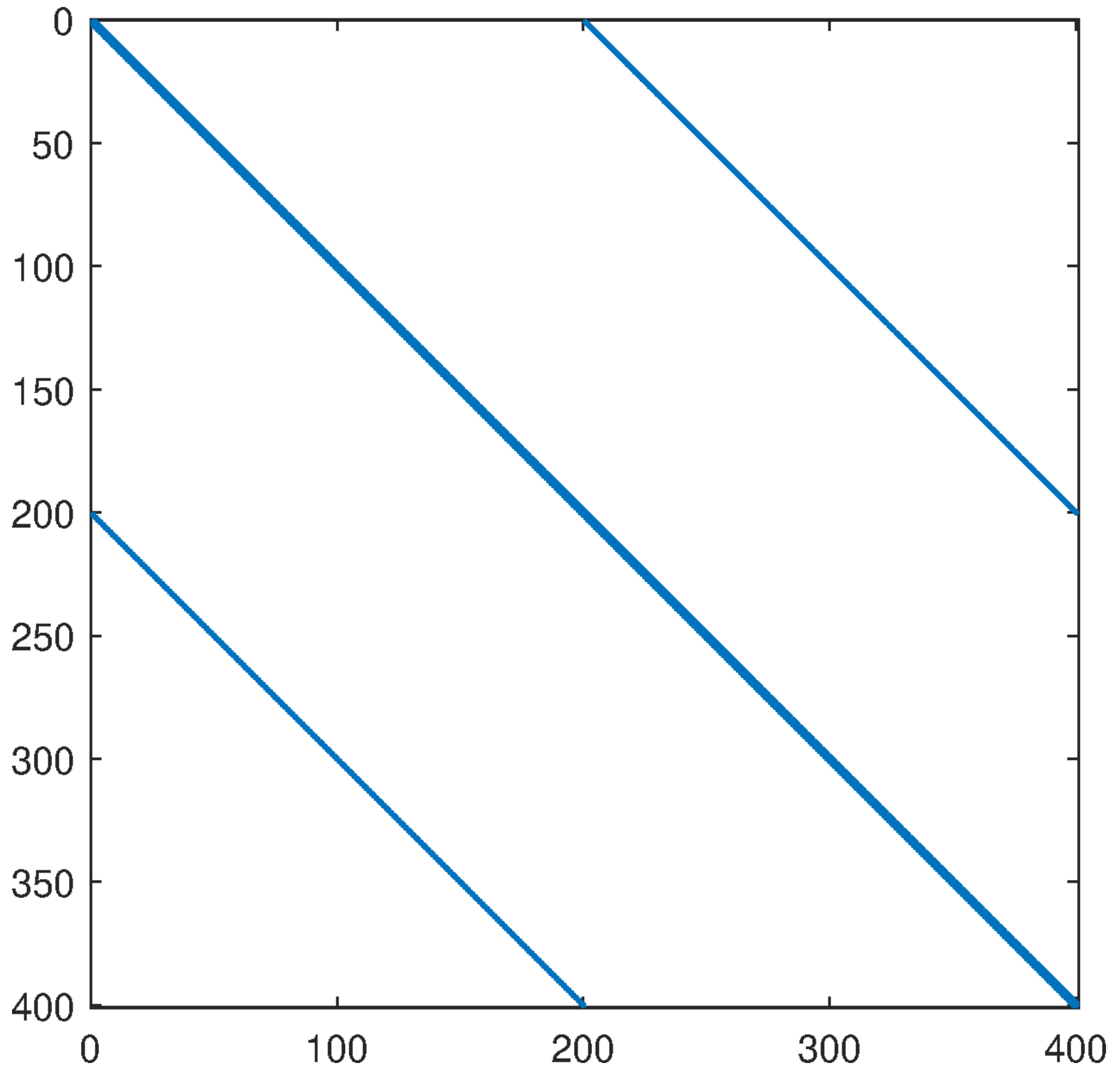
Notes:

1. Characteristics of system matrix to be solved.

2. Complexity of V2D code.

3. Earlier work on sparse approximate inverse preconditioners.

Resulting Matrix (if stored)



V2D Movie

Tests on Ookami

- Used Linux command **`perf stat -e duration_time –e cpu-cycles`**

- Used Gnu, Fujitsu, Cray, and Arm compilers

- Used optimization and non-optimization compiler features

- Also ran tests with and without SVE (vectorization).

- Used other analysis software, i.e., PAPI, Tau, and MAP.

Times by compiler

N_p	Direction		Times by Compiler (seconds)			
	NX1	NX2	GNU	Fujitsu	Cray (opt)	Cray (no-opt)
1	1	1	363.91	252.31	181.26	262.57
10	10	1	43.85	31.76	24.20	32.35
20	20	1	26.80	19.79	16.78	20.66
20	10	2	25.74	19.66	15.73	19.93
20	5	4	25.42	18.85	15.39	19.79
25	25	1	24.62	17.24	15.65	
40	40	1	25.30	13.97	19.12	
40	20	2	22.88	12.96	17.37	
40	10	4	21.91	13.04	17.16	
50	50	1	30.10	13.05	25.56	
50	25	2	29.26	12.09	24.07	
50	10	5	27.55	11.40	23.51	

Performance Application Programming Interface (PAPI)

- Interface and methodology for use of the performance counter hardware found in most major microprocessors

- Enables seeing, in near real time, the relation between processor performance and processor events

- Also provides access to a collection of components that expose performance measurement opportunities across the hardware and software stack

https://icl.utk.edu/papi/

PAPI Comments

- One can use PAPI with the Tau package (U Oregon) without any changes to the code. This can sample the code and the data produced can be viewed via Paraprof.

- One can add specific timing calls to the code to determine the time of sections. This can be done via Fortran modules (cf. Danny Vanpoucke's OOP Timer Example).

PAPI Comments Cont'd

Alternatively, one can merely insert the following two calls before and after a section of code being monitored:

```
call PAPIf_hl_region_begin('region-1', retval)
```

…*code that is being timed is included here.*

```
call PAPIf_hl_region_end('region-1', retval)
```

In either approach, an output file is produced which need to be processed to retrieve the various timing data.

Linear Algebra Routine Times

PAPI times (seconds)			
Routine	**No-SVE**	**SVE**	**SVE/No-SVE**
MATVEC	599	96	0.16
DPROD	132	24.3	0.18
DAXPY	206	53.8	0.26
DSCAL	153	47.7	0.31
DDAXPY	296	65	0.22

Conclusions

- Ookami is a tremendous resource for comparing/contrasting compilers.
- V2D is a *very* complicated piece of code.
- Varying the grid decomposition and the compiler result in different times of the runs.
- Using SVE on Ookami does produce a speedup with standard linear algebra operations, but the complexity of V2D does not exhibit as much of a speedup as would be expected.
- Further tests are needed.

THANK YOU FOR YOUR ATTENTION!

Vielen Dank!