

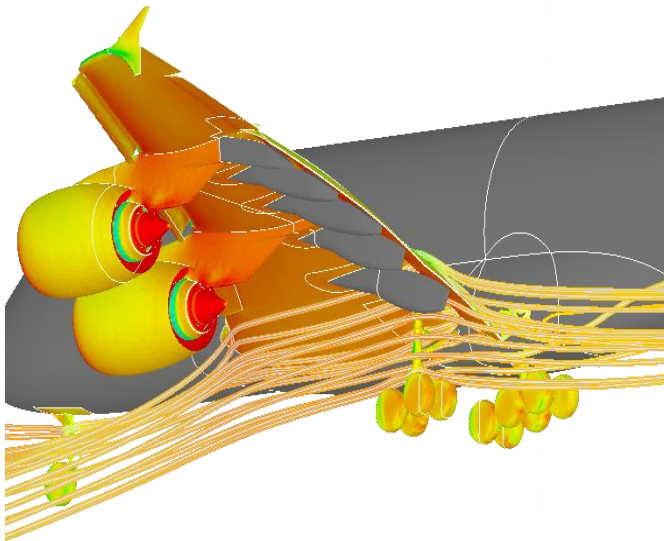
Petascale in CFD

Cetraro, 29th of June, 2011

Christian Simmendinger, T-Systems Solutions for Research

Overview

The TAU Solver



Scalability

- Core (Instruction Level)
 - SIMD – SRC-SRC
- Thread (Task Level)
 - Beyond OpenMP
- System Level
 - GASPI / PGAS API

- Maintainability of the Code
- Programmability
- Performance Portability

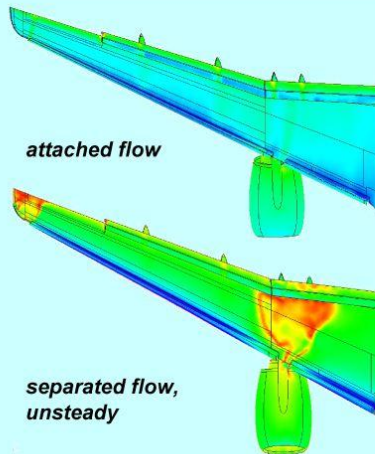
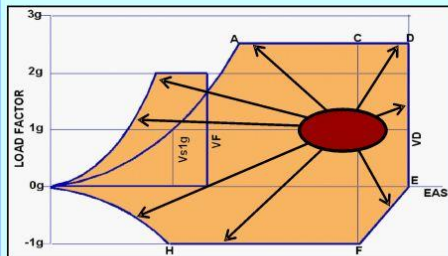
The TAU Solver

Vision: Digital Aircraft

TAU: 3D Finite Volume Reynolds Averaged Navier Stokes (RANS) Solver

Full flight envelope
coverage:

*CFD mostly done
near cruise point*



configurations:

clean



airbrakes deployed



high lift



- 50 flight points
- 100 mass cases
- 10 a/c configurations
- 5 maneuvers
- 20 gusts (gradient lengths)
- 4 control laws

~ 20,000,000 simulations

Engineering experience
for **current** configurations
and technologies

~ 100,000 simulations

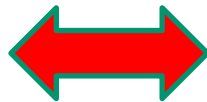
Programmability , Maintainability and Performance Portability: Instruction Level

SSE2/SSE4.2/AVX/MIC: From 128 bit SIMD to 512 bit SIMD.
16 DP Flop/Cycle – or just one.

- Statistical Fact:
Compiler performance doubles every 18 years. Do not expect compilers to resolve the SIMD issue for you. Especially C/C++ takes a severe performance hit due to aliasing.



SIMD Intrinsics

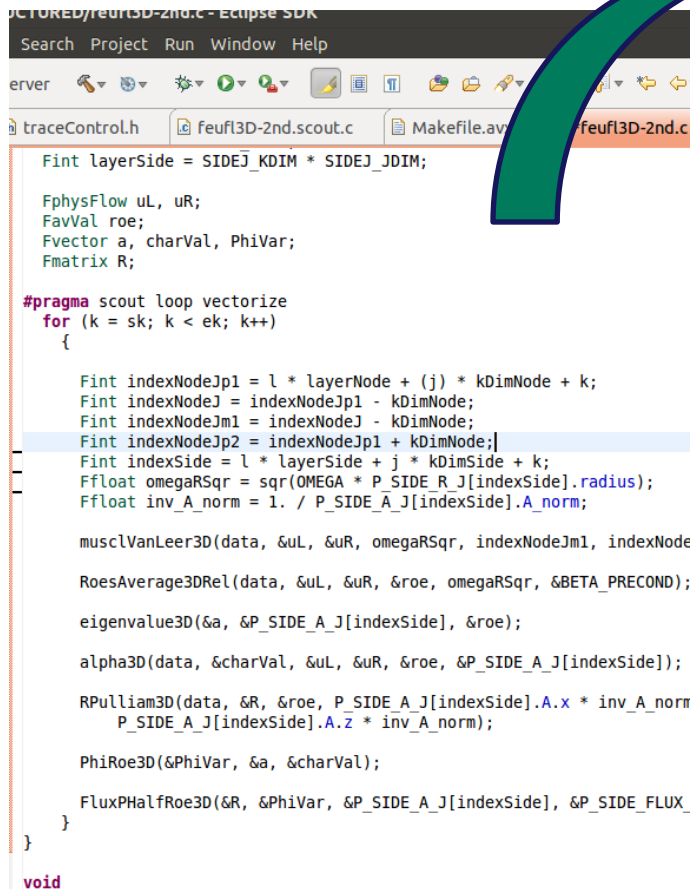


- Maintainability
- Programmability
- Performance-Portability

Scalability

- Core (Instruction Level)
 - SIMD – SRC-SRC
- Thread (Task Level)
 - Beyond OpenMP
- System Level
 - GASPI / PGAS API

Programmability , Maintainability and Performance Portability: Instruction Level



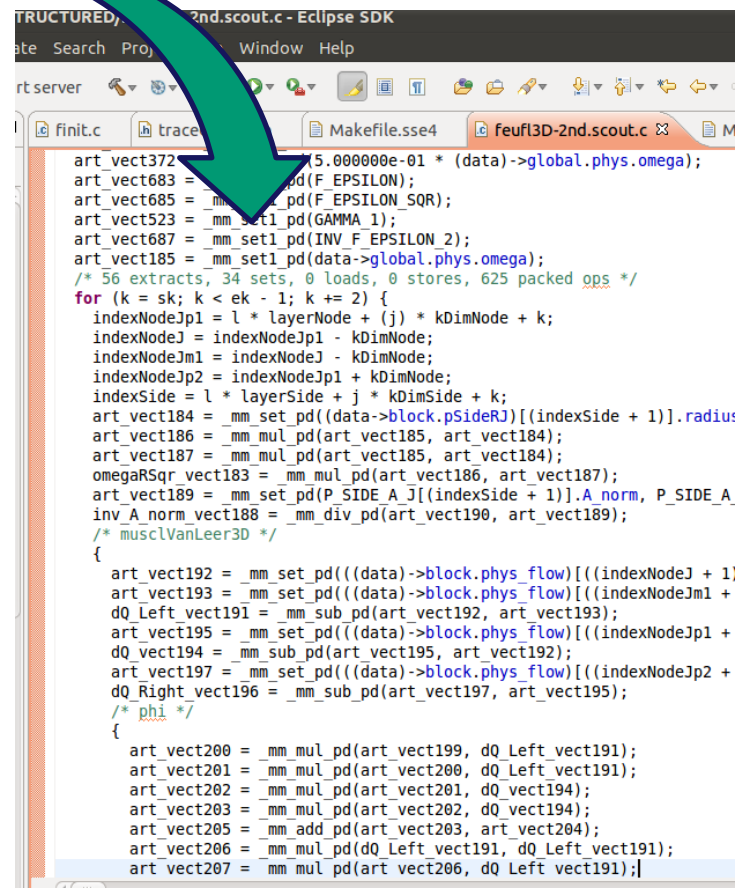
```
Fint layerSide = SIDEJ_KDIM * SIDEJ_JDIM;

FphysFlow uL, uR;
FavVal roe;
Fvector a, charVal, PhiVar;
Fmatrix R;

#pragma scout loop vectorize
for (k = sk; k < ek; k++)
{
    Fint indexNodeJp1 = l * layerNode + (j) * kDimNode + k;
    Fint indexNodeJ = indexNodeJp1 - kDimNode;
    Fint indexNodeJm1 = indexNodeJ - kDimNode;
    Fint indexNodeJp2 = indexNodeJp1 + kDimNode;
    Fint indexSide = l * layerSide + j * kDimSide + k;
    Ffloat omegaRSqr = sqr(OMEGA * P_SIDE_R_J[indexSide].radius);
    Ffloat inv_A_norm = 1. / P_SIDE_A_J[indexSide].A_norm;

    musclVanLeer3D(data, &uL, &uR, omegaRSqr, indexNodeJm1, indexNodeJ, indexNodeJp1, indexNodeJp2, &roe, omegaRSqr, &BETA_PRECOND);
    RoesAverage3DRel(data, &uL, &uR, &roe, omegaRSqr, &BETA_PRECOND);
    eigenvalue3D(&a, &P_SIDE_A_J[indexSide], &roe);
    alpha3D(data, &charVal, &uL, &uR, &roe, &P_SIDE_A_J[indexSide]);
    RPulliam3D(data, &R, &roe, P_SIDE_A_J[indexSide].A.x * inv_A_norm, P_SIDE_A_J[indexSide].A.z * inv_A_norm);
    PhiRoe3D(&PhiVar, &a, &charVal);
    FluxPHalfRoe3D(&R, &PhiVar, &P_SIDE_A_J[indexSide], &P_SIDE_FLUX_J[indexSide]);
}

void
```

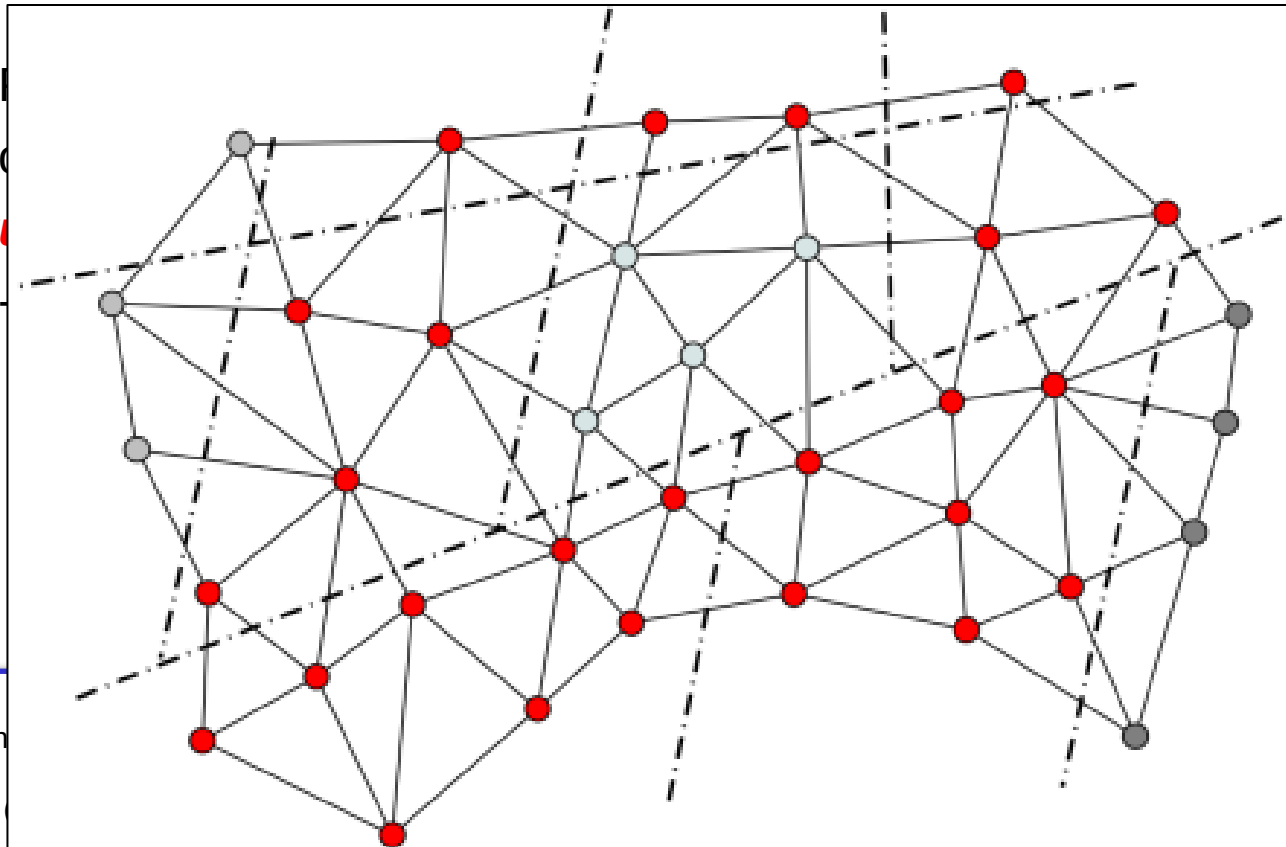


```
art_vect372 = _mm_set_pd(5.000000e-01 * (data->global.phys.omega);
art_vect683 = _mm_set_pd(F_EPSILON);
art_vect685 = _mm_set_pd(F_EPSILON_SQR);
art_vect523 = _mm_set1_pd(GAMMA_1);
art_vect687 = _mm_set1_pd(INV_F_EPSILON_2);
art_vect185 = _mm_set1_pd(data->global.phys.omega);
/* 56 extracts, 34 sets, 0 loads, 0 stores, 625 packed ops */
for (k = sk; k < ek - 1; k += 2) {
    indexNodeJp1 = l * layerNode + (j) * kDimNode + k;
    indexNodeJ = indexNodeJp1 - kDimNode;
    indexNodeJm1 = indexNodeJ - kDimNode;
    indexNodeJp2 = indexNodeJp1 + kDimNode;
    indexSide = l * layerSide + j * kDimSide + k;
    art_vect184 = _mm_set_pd((data->block.pSideRJ)[indexSide + 1].radius);
    art_vect186 = _mm_mul_pd(art_vect185, art_vect184);
    art_vect187 = _mm_mul_pd(art_vect185, art_vect184);
    omegaRSqr_vect183 = _mm_mul_pd(art_vect186, art_vect187);
    art_vect189 = _mm_set_pd(P_SIDE_A_J[indexSide + 1].A_norm, P_SIDE_A_J[indexSide + 1].A_norm);
    inv_A_norm_vect188 = _mm_div_pd(art_vect190, art_vect189);
    /* musclVanLeer3D */
    {
        art_vect192 = _mm_set_pd(((data->block.phys_flow)[indexNodeJ + 1].u, (data->block.phys_flow)[indexNodeJm1 + 1].u);
        dQ_Left_vect191 = _mm_sub_pd(art_vect192, art_vect193);
        art_vect195 = _mm_set_pd(((data->block.phys_flow)[indexNodeJp1 + 1].u, (data->block.phys_flow)[indexNodeJp2 + 1].u);
        dQ_Right_vect196 = _mm_sub_pd(art_vect197, art_vect195);
        /* phi */
        {
            art_vect200 = _mm_mul_pd(art_vect199, dQ_Left_vect191);
            art_vect201 = _mm_mul_pd(art_vect200, dQ_Left_vect191);
            art_vect202 = _mm_mul_pd(art_vect201, dQ_Left_vect191);
            art_vect203 = _mm_mul_pd(art_vect202, dQ_Left_vect191);
            art_vect205 = _mm_add_pd(art_vect203, art_vect204);
            art_vect206 = _mm_mul_pd(dQ_Left_vect191, dQ_Left_vect191);
            art_vect207 = _mm_mul_pd(art_vect206, dQ_Left_vect191);
        }
    }
}
```

Programmability , Maintainability and Performance Portability: Thread Level

From M

- Reso
- *Mutu*
- Sub-



Scalability

- Core (In
- Thread
- System Level
 - GASPI / PGAS API

Programmability , Maintainability and Performance Portability: Thread Level

- *Mutual Exclusion*

```
for (color = fcolor; color != NULL; color = color->succ)
    for (face = color->start; face < color->stop; face++)
```



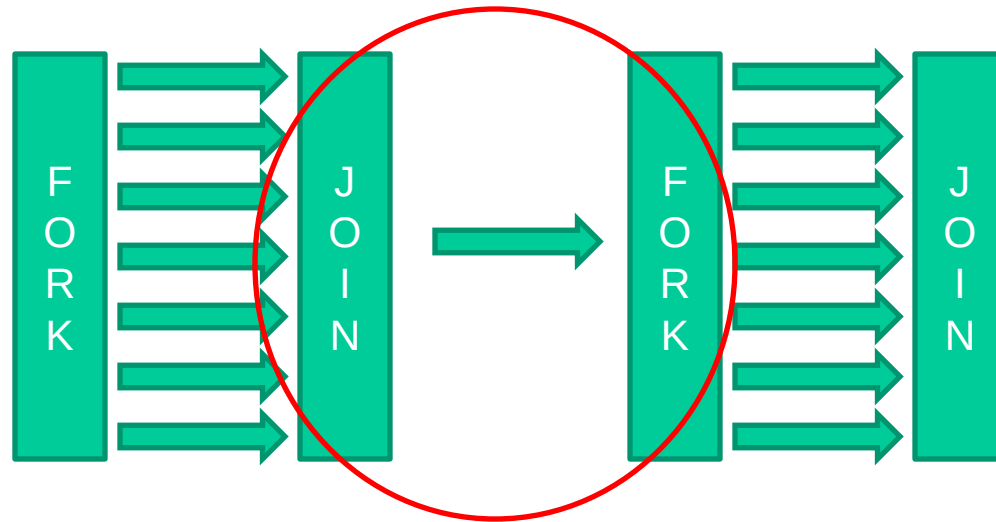
```
for(color=get_color();color != NULL; color=get_next_color(color))
    for (face = color->start; face < color->stop; face++)
```

- Maintainability
- Programmability
- Performance Portability

Programmability , Maintainability and Performance Portability: Thread Level

OpenMP – The Fork Join Model

- Incremental Parallelisation
- Amdahl Trap (Barrier and Load Imbalance)

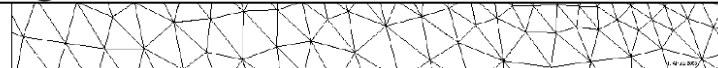
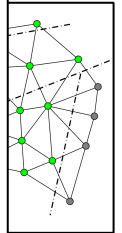
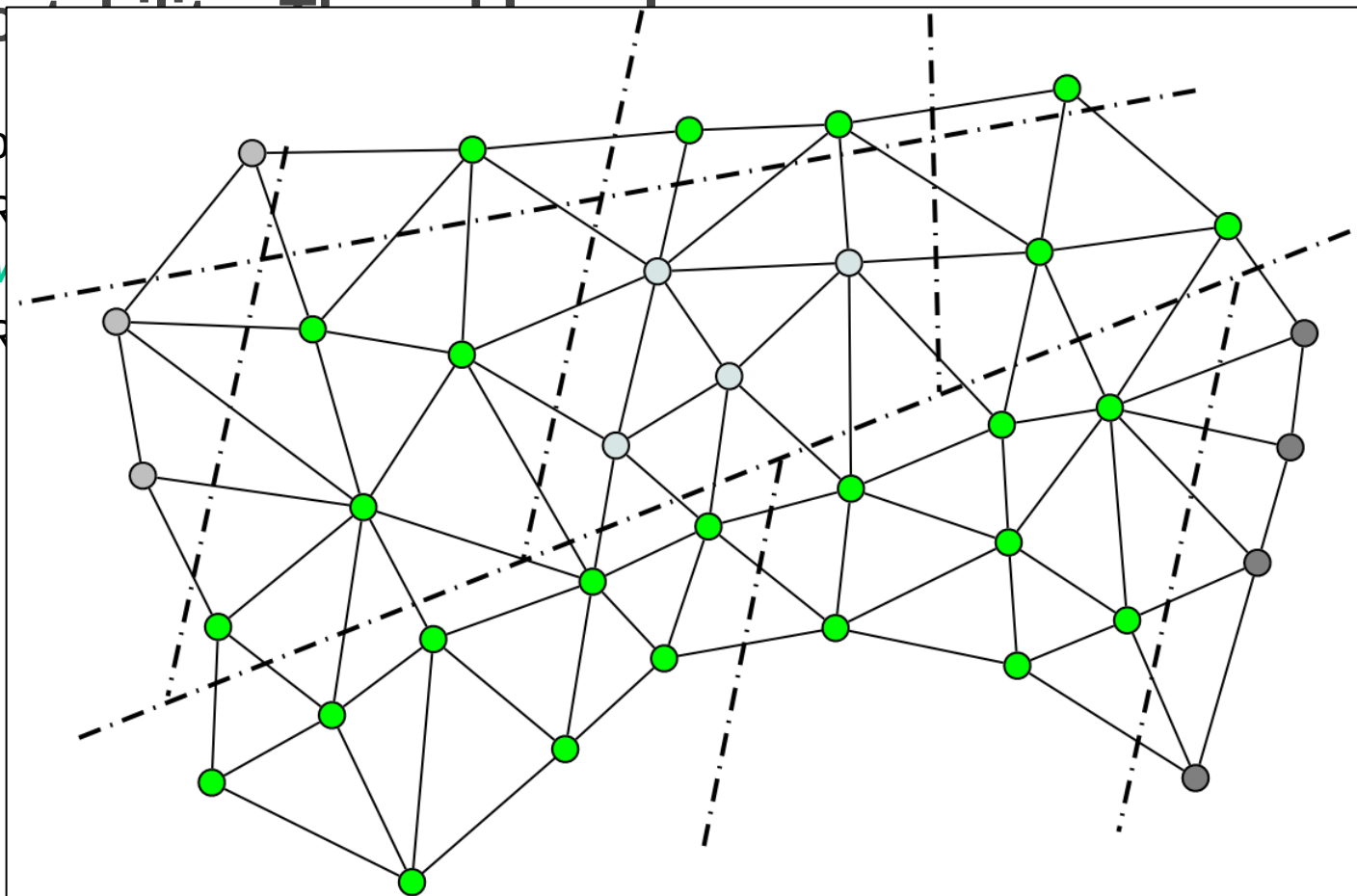


Programmability , Maintainability and Performance

Programmability = Maintainability

Beyond

- R
- M
- R



Programmability , Maintainability and Performance Portability: Thread Level

- *Mutual Completion*

```
for(color=get_color();color != NULL; color=get_next_color(color))  
    for (face = color->start; face < color->stop; face++)  
BARRIER;  
for(pnt = 0; pnt < nown; pnt++)
```



```
for(color=get_color();color != NULL; color=get_next_color(color))  
    for (face = color->start; face < color->stop; face++)  
while(get_pnt_range(&pstart, &pstop))  
    for (pnt = pstart; pnt < pstop; pnt++)...
```



- Maintainability
- Programmability
- Performance Portability

Programmability , Maintainability and Performance Portability: Thread Level

This Programming Model is *Generic*.

```
for(k = 0; k < KDIM; k++)  
    for (j = 0; j < JDIM; j++)  
        for (i = 0; i < IDIM; i++) {};  
BARRIER;  
for(k = 0; k < KDIM; k++) ...
```



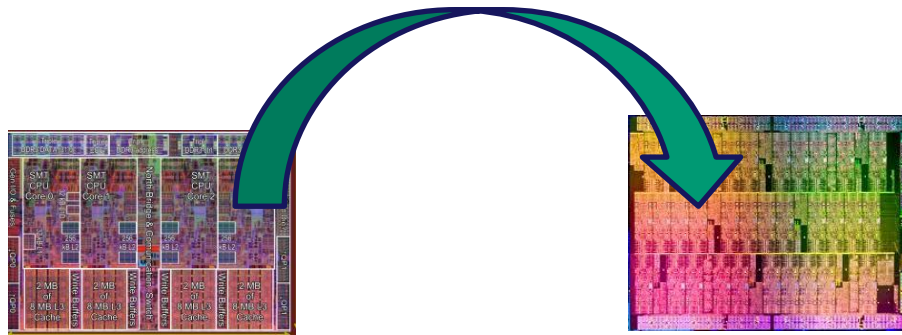
```
for(k = get_k_index(); k != -1; k = get_next_k_index(k))  
    for (j = 0; j < JDIM; j++)  
        for (i = 0; i < IDIM; i++) {};  
for(k = get_k_index(); k != -1; k = get_next_k_index(k)) ...
```

J.Jägersküpper (DLR), C.Simmendinger (T-Systems Sfr). A Novel Shared-Memory Thread-Pool Implementation for Hybrid Parallel CFD Solvers, EuroPar 2011, (to appear).

Programmability , Maintainability and Performance Portability: Thread Level

This extension of the OpenMP Fork-Join Programming Model is

- fully asynchronous
- data flow driven
- implicitly load balanced
- exclusively uses local locks instead of global barriers
- expected to scale to $O(10^2)$ cores



Programmability , Maintainability and Performance Portability: System Level

In a Partitioned Global Address Space every thread can read/write the entire global memory of the application.

- ➡ Opportunities for improved programmability
(PGAS languages like UPC, CAF, Chapel and others)
- ➡ Opportunities for improved *Scalability*
(GPI (Fraunhofer ITWM), GASPI)

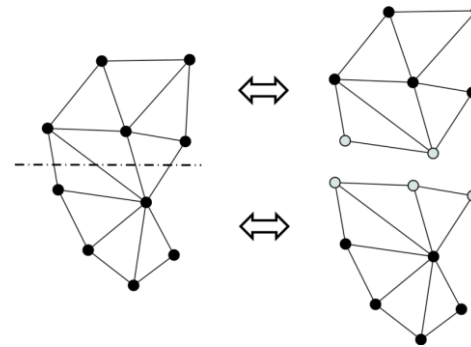
Global Address Space Programming Interface (GASPI)

A PGAS API which aims at replacing bulk-synchronous communication with asynchronous, 1-sided, RDMA driven communication patterns.

Scalability

- Core (Instruction Level)
 - SIMD – SRC-SRC
- Thread (Task Level)
 - Beyond OpenMP
- System Level
 - GASPI / PGAS API

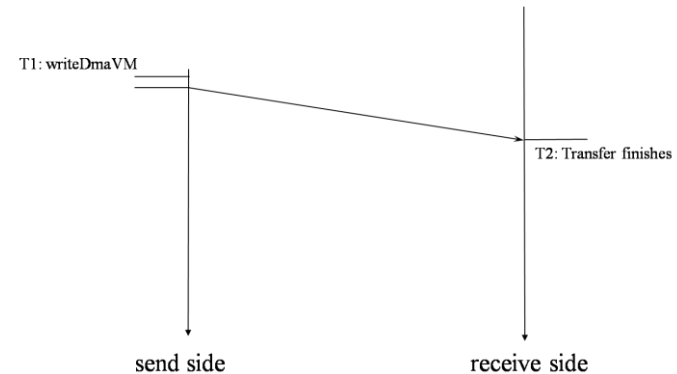
Programmability , Maintainability and Performance Portability: System Level



Overlapping Communication and Computation

```
if (this_thread_exchanges_pnt_data(g)) /* lock halo */
{
    exchange_pointdata();
    post_exch_pnt_data(); /* unlock halo*/
}
```

Programmability , Maintainability and Performance Portability: System



Multithreaded send/recv

```
if ((sid = get_seq_num_and_lock(g)) < num_send_threads(g))
    send_dbl_threaded(comap, comap->ncommdomains, comap->commpartner,
        data, dim2, key);

If ((sid = thread_seq_num()) < num_recv_threads(g))
    recv_dbl_threaded(g, num_recv_threads, sid,
        comap, comap->ncommdomains, comap->commpartner, data, dim2, key);
```

Programmability , Maintainability and Performance Portability: System Level

Strong Scaling Use Case: **How far can we scale ?**

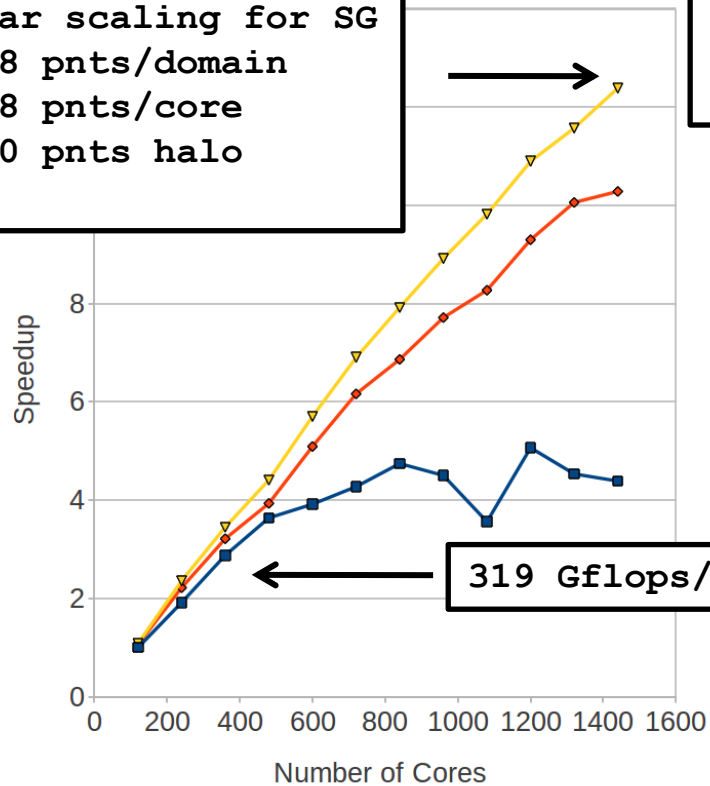
4W Multigrid, 1 Iteration Step.

- 1 Allreduce
- 140 Next Neighbour Halo Exchanges



F6, SG, 2,000,000 Points

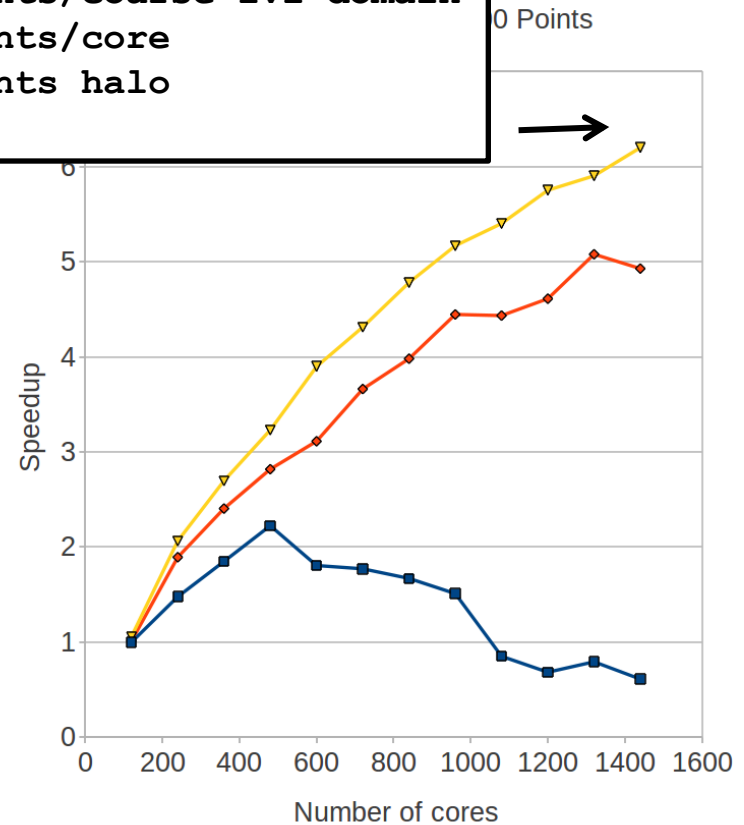
Linear scaling for SG
8328 pnts/domain
1388 pnts/core
+4000 pnts halo



319 Gflops/sec

■ MPI 2009.2.0
▼ GPI/OpenMP
◆ MPI/OpenMP

80 pnts/coarse lvl domain
13 pnts/core
+160 pnts halo

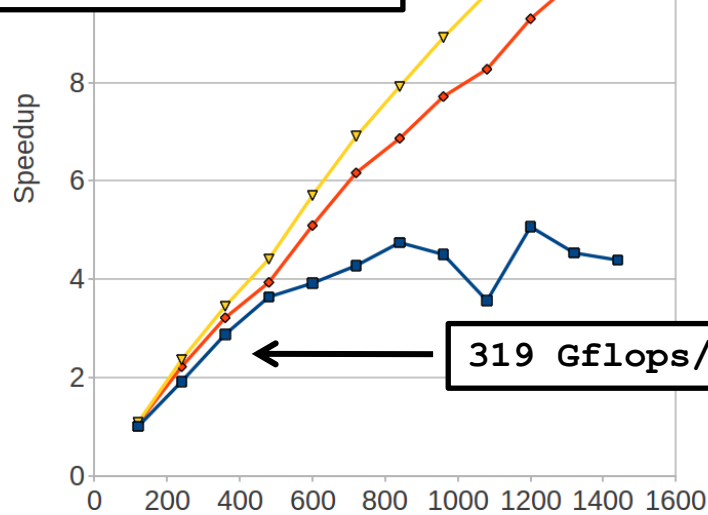


■ MPI 2009.2.0
▼ GPI/OpenMP
◆ MPI/OpenMP

F6, SG, 2,000,000 Points

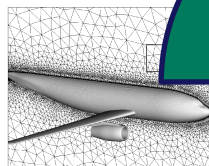
x 30

Linear scaling for SG
8328 pnts/domain
1388 pnts/core
+4000 pnts halo



319 Gflops/sec

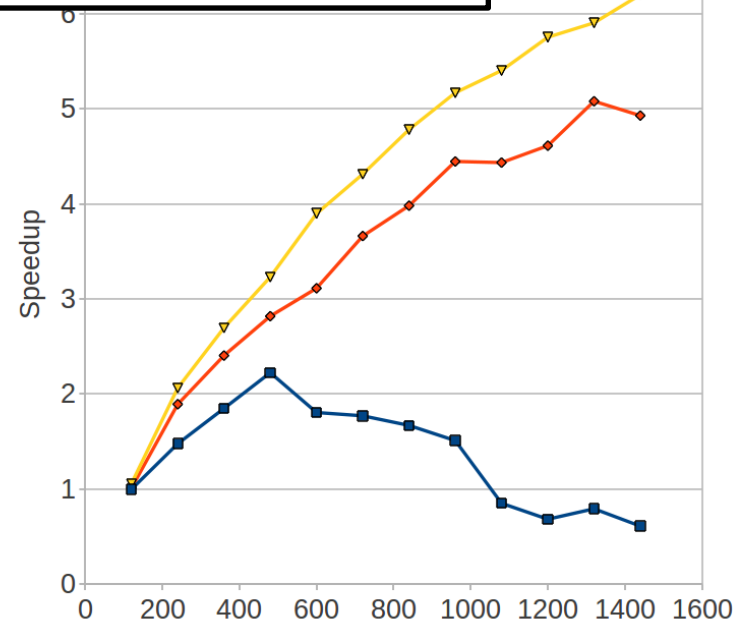
Number of Processors x 30



MPI 2009.2.0
MPI/OpenMP
MPI/OpenMP

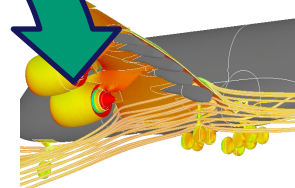
80 pnts/coarse lvl domain
13 pnts/core
+160 pnts halo

2,000,000 Points

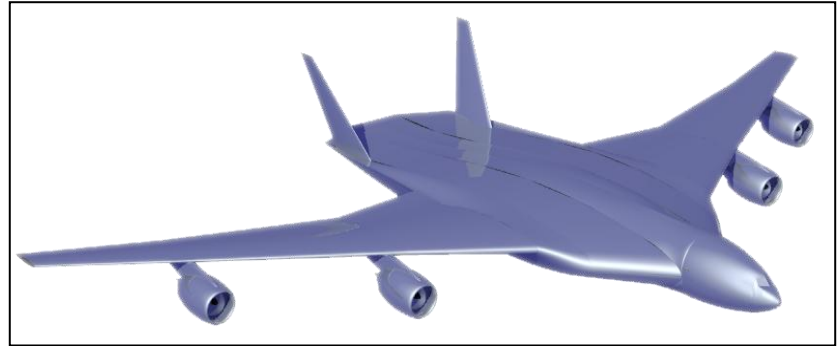


Number of Processors x 30

MPI 2009.2.0
MPI/OpenMP
MPI/OpenMP

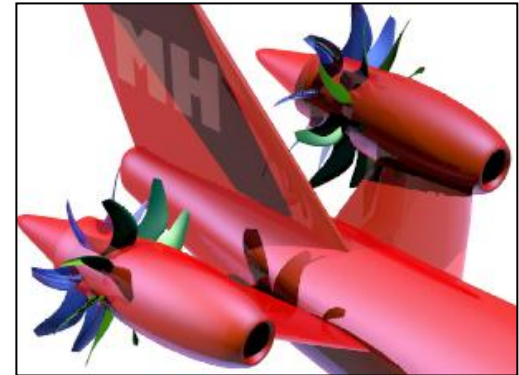


Summary



Scalability

- Core (Instruction Level)
 - SIMD – SRC-SRC
- Thread (Task Level)
 - Beyond OpenMP
- System Level
 - GASPI / PGAS API / GASPI



We are confident, that we can reach O(Petascale) within the constraints of a typical production run.