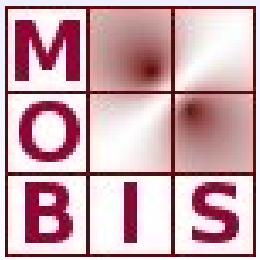
High Performance Computing with OpenACC
HPC II

Stefan Rosenberger

Supervisor: Univ.-Prof. Dipl.-Ing. Dr. Gundolf Haase
Institute of Mathematics and Scientific Computing
Universität Graz
SFB Research Center MOBIS

June 28, 2017



1 Introduction

2 Redesign

3 How to use OpenACC

4 Coalescing for OpenACC

5 Schön

Coding vs. Coding

Source code version 1:

```
#include <iostream>
void* voidfunction(void* (*funpoint)(void*), void* param) {
    return funpoint(param);
}
void* outthis(void* str) {
    static int ret = false;
    if (ret = (str != NULL))
        std::cout << (char*) str;
    return (void*) (int*) &ret;
}

int main() {
    return *(int*) voidfunction(&outthis,
        (void*)"\x48\x65\x6c\x6c\x6f\x20\x57\x6f\x72\x6c\x64");
}
```

Coding vs. Coding

Source code version 2:

```
int main()
{
    std::cout << "Hello World";
}
```

Coding vs. Coding

Source code version 1 again:

```
#include <iostream>
void* voidfunction(void* (*funpoint)(void*), void* param) {
    return funpoint(param);
}

void* outthis(void* str) {
    static int ret = false;
    if (ret = (str != NULL))
        std::cout << (char*) str;
    return (void*) (int*) &ret;
}

// This Code creates the output "Hello World"
int main() {
    return *(int*) voidfunction(&outthis,
        (void*)"\x48\x65\x6c\x6c\x6f\x20\x57\x6f\x72\x6c\x64");
}
```

Conclusion you should take from this Talk

- *Clean* coding is required in HPC!
- Documentation is not a possible work, it is MANDATORY!
- Clean up of the CODE is not a possible work, it is MANDATORY!
- Documentation improves CODE, since we have a chance to understand the CODE.

1 Introduction

2 Redesign

3 How to use OpenACC

4 Coalescing for OpenACC

5 Schön

Old Interface

I had to start with code like this:

```
void AMG1_J(int k)
{
    int l = k++;
    if(l < _level)
    {
#ifdef OPENMP
        simple_prolongation<T, S> _P(_rcnt[l], _rdsp[l], _rcol[l]);
        linear_operator<T, S> _R(_scnt[l], _sdsp[l], _scol[l], _sele[l]);
#else
        simple_prolongation<T, S> _P(_rcnt[l], _rcol[l]);
        simple_restriction<T, S> _R(_rcnt[l], _rcol[l]);
#endif
        jacobi<T, S> _J(_acnt[l], _adsp[l], _acol[l], _aele[l], _adia[l], _v[l], _omega,
             _com[l]);
        for(int i=0; i<_nsmooth; i++) _J(_f[l], _u[l]);
        residual(_acnt[l], _adsp[l], _acol[l], _aele[l], _f[l], _u[l], _r[l]);
        _R(_r[l], _f[l + 1]);

        AMG1_J(k);

        _P(_u[l + 1], _s[l]);
        amg_solver<T, S>::add(_s[l], _u[l]);
        for(int i=0; i<_nsmooth; i++) _J(_f[l], _u[l]);
    }
    else
    {
        (*_inv)(_f[l], _u[l]);
    }
}
```

Notes on the Old Interface

- **Every** element for the constructors exist **before** the function call!
- Basic Idea of the CODE: it should be flexible with respect to the operators!
- Therefore, none of the members can be set as **const**.

Notes on the Old Interface

- **Every** element for the constructors exist **before** the function call!
- Basic Idea of the CODE: it should be flexible with respect to the operators!
- Therefore, none of the members can be set as **const**.
- Next *worker* for the project had to implement the code in a different project.
- No documentation was written, therefore **every useful ansatz was lost**.

Ansatz for the improvement

- **First:** I document what I have done!!!!!!
- **Second:** Every kernel is written *with respect to the matrix*, therefore I save the elements **and** the routines together!
- **And last, but not least:** const, const, const, const, ...

Ansatz for the improvement

- **First:** I document what I have done!!!!!!
- **Second:** Every kernel is written *with respect to the matrix*, therefore I save the elements **and** the routines together!
- **And last, but not least:** const, const, const, const, ...
- Out of this ansatz, I've decided to write a basis class **toolbox_matrix** for my calculations.
- Every new *matrix*-routine is now written as a subclass of **toolbox_matrix**.
- One class for the constuctor: **CRSmatrix**.

The class CRSmatrix

We use the compact row storage format:

```
template<class T, class S>
class CRSmatrix: public toolbox_matrix<T, S> {
private:
        toolbox_vector<T> _nod;

        toolbox_vector<T> _cnt;
        toolbox_vector<T> _col;
        toolbox_vector<S> _ele;
        toolbox_vector<T> _dsp;

        toolbox_vector<S> _dia;

        int _num_row;

        // This routine calculates _v = A * _u
        void matrix_vector(const toolbox_vector<S> &_u, toolbox_vector<S> &_v,
                           const int _start, const int _stop) const;

        // This routine calculates _v = _v + A * _u
        void matrix_vector_add(const toolbox_vector<S> &_u, toolbox_vector<S> &_v,
                           const int _start, const int _stop) const;
```

The class CRSmatrix

We use the compact row storage format:

```
template<class T, class S>
class CRSmatrix: public toolbox_matrix<T, S> {
private:
        toolbox_vector<T> _nod;

        toolbox_vector<T> _cnt;
        toolbox_vector<T> _col;
        toolbox_vector<S> _ele;
        toolbox_vector<T> _dsp;

        toolbox_vector<S> _dia;

        int _num_row;

        // This routine calculates _v = A * _u
        void matrix_vector(const toolbox_vector<S> &_u, toolbox_vector<S> &_v,
                           const int _start, const int _stop) const;

        // This routine calculates _v = _v + A * _u
        void matrix_vector_add(const toolbox_vector<S> &_u, toolbox_vector<S> &_v,
                           const int _start, const int _stop) const;
```

I want to show the application on the Jacobi algorithm!

Jacobi Old

We use the compact row storage format:

```
template<class T, class S>
class jacobi_old {
private:
   const toolbox_vector<T> &_acnt;
   const toolbox_vector<T> &_adsp;
   const toolbox_vector<T> &_acol;
   const toolbox_vector<S> &_aele;
   const toolbox_vector<S> &_adia;
   toolbox_vector<S> &_v;
   const S _omega;
   communicator<T, S> &_com;

   void iterate(const toolbox_vector<S> &_f, const toolbox_vector<S> &_u,
         int start, int stop) {
      if (stop == 0)
         return;

      const T *__restrict acnt = _acnt.data(), *__restrict acol =
            _acol.data();
      const S *__restrict aele = _aele.data();
      const S *__restrict f = _f.data();
      const S *__restrict u = _u.data();
      S *__restrict v = _v.data();

      const T *__restrict adsp = _adsp.data();
```

Jacobi Old

We use the compact row storage format:

```
#ifdef OPENMP
#pragma omp parallel for schedule(guided, 2)
      for (int i = start; i < stop; i++) {
         S s = f[i];
         const T *p_col = acol + adsp[i];
         const S *p_ele = aele + adsp[i];
         T csize = acnt[i];

         for (T j = 0; j < csize; j++) {
            T c = p_col[j];
            S t = p_ele[j];
            s -= t * u[c];
         }
         v[i] = s;

      }
#else
      const T *p_col = acol + adsp[start];
      const S *p_ele = aele + adsp[start];
      for (int i = start; i < stop; i++) {
         S s = f[i];
         T j = 0, csize = acnt[i];

         while (j++ != csize) {
            T c = *p_col++;
            S t = *p_ele++;
```

Jacobi Old

I have tested it: I would need more than 10 slides to show it .....

Jacobi New

```
template<class T, class S>
class jacobi: public inverse_operator<T, S> {
private:
   const toolbox_matrix<T, S> &_A;
   toolbox_vector<S> _v;
   const S _omega;
   communicator<T, S> &_com;

   const bool prepare_for_device;
   int _nshared, _network_size;

   void iterate(const toolbox_vector<S> & _f, const toolbox_vector<S> & _u,
         int start, int stop) {
      _A.Residual(_f, _u, _v, start, stop); // _v = _f - _A*_u
   }

   void iterate(const toolbox_vector<S> & _f, const toolbox_vector<S> & _u) {
      _A.Residual(_f, _u, _v); // _v = _f - _A*_u
   }

   void z_iterate(const toolbox_vector<S> &_f, toolbox_vector<S> &_u,
         int start, int stop) const {
      _A.DiagonalVectorScaled(_f, _u, _omega, start, stop); // _u = _omega * D^{-1}*_f
   }

   void z_iterate(const toolbox_vector<S> &_f, toolbox_vector<S> &_u) const {
      _A.DiagonalVectorScaled(_f, _u, _omega); // _u = _omega * D^{-1}*_f
   }
```

Jacobi New

```
public:
   // Constructor
   jacobi(const toolbox_matrix<T, S> &A, communicator<T, S> & com, const S omega) :
         _A(A), _v(A.getrow(), 0.0), _omega(omega), _com(com), _prepare_for_device(
               A.IsDeviceMethode()) {
      network::size(_network_size);
      _nshared = _com.shared_nodes().size();

      if (_prepare_for_device) {
#pragma acc enter data pcopyin(this[0:1])
         _v.todev();
      }
   }
   // Destructor
   ~jacobi() {
      if (_prepare_for_device) {
         _v.fromdev();
#pragma acc exit data delete(this[0:1])
      }
   };
   // Operator to calculate _u = _u + _omega*D^{-1}*(_f - _A*_u)
   void operator()(const toolbox_vector<S> &_f, toolbox_vector<S> &_u) {
#ifdef PT_OVERLAP_JACOBI
      // Asynchron communication for MPI>1
      const int dsize = _f.size();

      int rank;
      network::rank(rank);
      // WARNING: for dpn>1, we need to redefine the "first" and "second" variable!!
```

Jacobi New

```
      const int first = (_nshared/_block_size +1) * _block_size;
      const int second = (( (first + dsize)/2 )/_block_size) *_block_size;

      iterate(_f, _u, 0, first);
      if(_prepare_for_device && _network_size > 1) _v.updatehost_shared_nodes(_nshared);
      _com.accumulate_start(_v); // Initialize communication
      iterate(_f, _u, first, second);
      _com.accumulate_compute(_v); // Send data
      iterate(_f, _u, second, dsize);
      _com.accumulate_finish(_v); // Recive data

      if(_prepare_for_device && _network_size > 1) _v.updatedev_shared_nodes(_nshared);
#else
      iterate(_f, _u);

      if(_prepare_for_device && _network_size > 1) _v.updatehost_shared_nodes(_nshared);
      _com.accumulate(_v);
      if(_prepare_for_device && _network_size > 1) _v.updatedev_shared_nodes(_nshared);
#endif
      _A.DiagonalVectorScaledAdd(_v, _u, _omega);
   }

   // Operator to calculate _u = _omega * D^{-1}*_f
   void zero_iteration(const toolbox_vector<S> &_f, toolbox_vector<S> &_u) {
#ifdef PT_OVERLAP_JACOBI
      // Asynchron communication for MPI>1
      const int dsize = _A.getrow();
```

Jacobi New

```
        const int first = (_nshared/_block_size +1) * _block_size;
        const int second = (( (first + dsize)/2 )/_block_size) *_block_size;

        z_iterate(_f, _u, 0, first);
        if(_prepare_for_device && _network_size > 1) _u.updatehost_shared_nodes(_nshared);

        _com.accumulate_start(_u); // Initialize communication
        z_iterate(_f, _u, first, second);
        _com.accumulate_compute(_u); // Send data
        z_iterate(_f, _u, second, dsize);
        _com.accumulate_finish(_u); // Recive data

        if(_prepare_for_device && _network_size > 1) _u.updatedev_shared_nodes(_nshared);
#else
        z_iterate(_f, _u);

        if(_prepare_for_device && _network_size > 1) _u.updatehost_shared_nodes(_nshared);
        _com.accumulate(_u);
        if(_prepare_for_device && _network_size > 1) _u.updatedev_shared_nodes(_nshared);
#endif
    }
};
```

Jacobi New

We have now the following advantage:

- It is much easier to understand the algorithm!
- We have a **much higher** functionality: Device/Host; overlapped for Device **and** Host!
- If one want to use a different type of matrix format, one can simply write new matrix-class!
- We have constant methods in matrix an jacobi algorithm!

Compare different versions on the Host

We calculate the solution for the TBunnyC mesh on the host:

(We calculate the median of 3 runs!)

CG with Jacobi						
		1 CPU	1 Thr.	2 Thr.	4 Thr.	8 Thr.
PGI 17.1	old:	10,041s	10,944s	6,557s	5,372s	5,569s
	new:	9.810s	10,342s	6,114s	4,978s	5,180s
	diff:	+2,35%	+5,82%	+7,24%	+7,91%	+7,51%
Intel 1700	old:					
	new:					
	diff:					

Measuring error (approximately:) $\pm 0,05$s

Compare different versions on the Host

We calculate the solution for the TBunnyC mesh on the host:

(We calculate the median of 3 runs!)

CG with Jacobi						
		1 CPU	1 Thr.	2 Thr.	4 Thr.	8 Thr.
PGI 17.1	old:	10,041s	10,944s	6,557s	5,372s	5,569s
	new:	9.810s	10,342s	6,114s	4,978s	5,180s
	diff:	+2,35%	+5,82%	+7,24%	+7,91%	+7,51%
Intel 1700	old:	9,641s	9,807s	6,114s	5,287s	5,431s
	new:	9,187s	9,383s	5,762s	4,933s	5,08s
	diff:	+4,94%	+4,52%	+6,11%	+7,18%	+8,94%

Measuring error (approximately:) $\pm 0,05$s

Compare different versions on the Host

We calculate the solution for the TBunnyC mesh on the host:

(We calculate the median of 3 runs!)

CG with AMG						
		1 CPU	1 Thr.	2 Thr.	4 Thr.	8 Thr.
PGI 17.1	old:	2,732s	2,735s	1,592s	1,281s	1,338s
	new:	2,539s	2,717s	1,545s	1,227s	1,281s
	diff:	+7,60%	+0,66%	+3,04%	+4,40%	+4,44%
Intel 1700	old:					
	new:					
	diff:					

Measuring error (approximately:) $\pm 0,05$s

Compare different versions on the Host

We calculate the solution for the TBunnyC mesh on the host:

(We calculate the median of 3 runs!)

CG with AMG						
		1 CPU	1 Thr.	2 Thr.	4 Thr.	8 Thr.
PGI 17.1	old:	2,732s	2,735s	1,592s	1,281s	1,338s
	new:	2,539s	2,717s	1,545s	1,227s	1,281s
	diff:	+7,60%	+0,66%	+3,04%	+4,40%	+4,44%
Intel 1700	old:	2,469s	2,476s	1,486s	1,244s	1,309s
	new:	2,363s	2,383s	1,431s	1,195s	1,242s
	diff:	+4,49%	+3,90%	+3,84%	+4,10%	+5,39%

Measuring error (approximately:) $\pm 0,05$s

Compare different versions on the Host

We calculate the solution a 4 times - TBunnyC -size mesh on the host:
(We use only one run!)

CG with Jacobi						
		1 CPU	1 Thr.	2 Thr.	4 Thr.	8 Thr.
PGI 17.1	old:	55,365s	55,290s	33,977s	29,181s	30,381s
	new:	47,850s	50,099s	31,822s	27,800s	28,986s
	diff:	+15,70%	+10,36%	+6,77%	+4,93%	+4,81%
Intel 1700	old:					
	new:					
	diff:					

Measuring error (approximately:) $\pm 0,05$s

Compare different versions on the Host

We calculate the solution a 4 times - TBunnyC -size mesh on the host:
(We use only one run!)

CG with Jacobi						
		1 CPU	1 Thr.	2 Thr.	4 Thr.	8 Thr.
PGI 17.1	old:	55,365s	55,290s	33,977s	29,181s	30,381s
	new:	47,850s	50,099s	31,822s	27,800s	28,986s
	diff:	+15,70%	+10,36%	+6,77%	+4,93%	+4,81%
Intel 1700	old:	51,992s	50,159s	32,148s	28,672s	29,914s
	new:	48,949s	48,803s	31,494s	27,818s	28,736s
	diff:	+6,22%	+2,78%	+2,08%	+3,07%	+4,10%

Measuring error (approximately:) $\pm 0,05$s

Compare different version on the Host

We calculate the solution a 4 times - TBunnyC -size mesh on the host:
(We use only one run!)

CG with AMG						
		1 CPU	1 Thr.	2 Thr.	4 Thr.	8 Thr.
PGI 17.1	old:	78,797s	74,735s	44,764s	37,675s	38,662s
	new:	66,460s	70,821s	42,867s	36,480s	37,444s
	diff:	+18,56%	+5,53%	+4,43%	+3,27%	+3,25%
Intel 1700	old:					
	new:					
	diff:					

Measuring error (approximately:) $\pm 0,05$s

Compare different version on the Host

We calculate the solution a 4 times - TBunnyC -size mesh on the host:
(We use only one run!)

CG with AMG						
		1 CPU	1 Thr.	2 Thr.	4 Thr.	8 Thr.
PGI 17.1	old:	78,797s	74,735s	44,764s	37,675s	38,662s
	new:	66,460s	70,821s	42,867s	36,480s	37,444s
	diff:	+18,56%	+5,53%	+4,43%	+3,27%	+3,25%
Intel 1700	old:	69,431s	66,228s	42,227s	36,808s	37,934s
	new:	64,806s	65,444s	41,735s	36,007s	36,826s
	diff:	+7,14%	+1,20%	+1,18%	+2,22%	+3,01%

Measuring error (approximately:) $\pm 0,05$s

Conclusion for the Host-Tests

- In **ANY** case is it useful to write *correct* CODE.
- Dependent on the problem, we can get **up to** 20% speed up (in particular for the PGI-compiler)!
- We will show now, that this *style to code* is **in particular** with OpenACC important!

1 Introduction

2 Redesign

3 How to use OpenACC

4 Coalescing for OpenACC

5 Schön

How to parallelize with OpenACC

I've parallelized the following code:

```
for(int i = 0; i < size; i++)
{
   int c = rcnt[i];
   switch(c)
   {
   case 0:
      //cout << "*";
      break;
   case 1:
      s[i] = u[*rcol++];
      break;
   case 2:
      t = u[*rcol++];
      t += u[*rcol++];
      s[i] = t / S(2.0);
      break;
   default:
      t = S(0.0);
      for(int j = 0; j < c; j++)
      {
         t += u[*rcol++];
      }
      s[i] = t / S(c);
      break;
   }
}
```

How to parallelize with OpenACC

For OpenACC one can consider the parallelization for the most important kernel:

```
#pragma acc parallel loop pcopy(...) pcopyin(....)
for (int i = 0; i < num_row; i++) {
    const T *__restrict pcol = col + dsp[i];
    const T c = cnt[i];

    S t = S(0.0);
#pragma acc loop seq
    for (int j = 0; j < c; j++) {
        t += u_coarse[*pcol++];
    }
    u_fine[i] += t / S(c);
}
```

How to parallelize with OpenACC

Two days search for a BUG later:

```
#pragma acc parallel loop pcopy(...) pcopyin(....)
for (int i = 0; i < num_row; i++) {
    const T *__restrict pcol = col + dsp[i];
    const T c = cnt[i];
    switch (c) {
    case 0:
        // In the case of MPI>1 it might happen, that c = 0!!!!
        break;
    default:
        S t = S(0.0);
#pragma acc loop seq
        for (int j = 0; j < c; j++) {
            t += u_coarse[*pcol++];
        }
        u_fine[i] += t / S(c);
        break;
    }
}
```

PGI-Bugreport Nr.: FS24002

- I've created a optimized OpenACC parallel version of our CODE!
- Comparision with the existing CUDA version:
  - With Jacobi: CUDA was 2,5 times faster than OpenACC!
  - With AMG: CUDA was 1,7 times faster than OpenACC!

PGI-Bugreport Nr.: FS24002

- I've created a optimized OpenACC parallel version of our CODE!
- Comparision with the existing CUDA version:
  - With Jacobi: CUDA was 2,5 times faster than OpenACC!
  - With AMG: CUDA was 1,7 times faster than OpenACC!
- I used the environment variable *export PGI_ACC_NOTIFY=3* to find all data transfer points to the device!
- OpenACC uses *unified memory* for every scalar (the user can not handle them direct/easily).
- I analyzed the code with the nvcc profiler.

PGI-Bugreport Nr.: FS24002

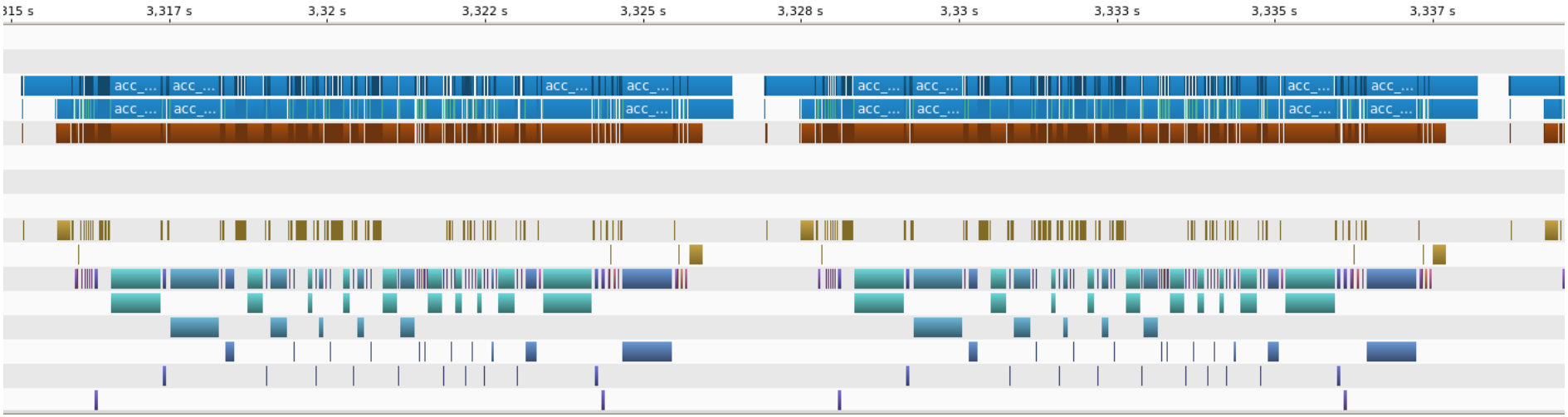


Figure: Gaps between kernels (AMG on Nvidia-Pascal).

PGI-Bugreport Nr.: FS24002

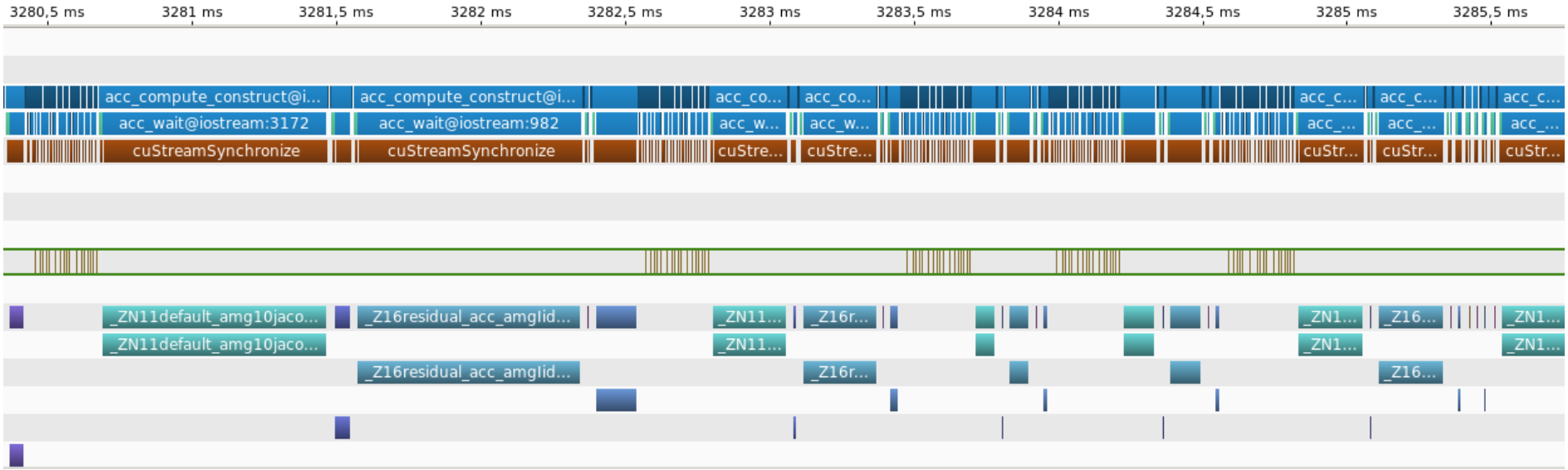


Figure: Pointer synchronisation host $\longleftrightarrow$ device. (AMG on Nvidia-Pascal)

The reason

We had this:

```
void AMG1_J(int k)
{
    int l = k++;
    if(l < _level)
    {
#ifdef OPENMP
        simple_prolongation<T, S> _P(_rcnt[l], _rdsp[l], _rcol[l]);
        linear_operator<T, S> _R(_scnt[l], _sdsp[l], _scol[l], _sele[l]);
#else
        simple_prolongation<T, S> _P(_rcnt[l], _rcol[l]);
        simple_restriction<T, S> _R(_rcnt[l], _rcol[l]);
#endif
        jacobi<T, S> _J(_acnt[l], _adsp[l], _acol[l], _aele[l], _adia[l], _v[l], _omega,
             _com[l]);
        for(int i=0; i<_nsmooth; i++) _J(_f[l], _u[l]);
        residual(_acnt[l], _adsp[l], _acol[l], _aele[l], _f[l], _u[l], _r[l]);
        _R(_r[l], _f[l + 1]);

        AMG1_J(k);

        _P(_u[l + 1], _s[l]);
        amg_solver<T, S>::add(_s[l], _u[l]);
        for(int i=0; i<_nsmooth; i++) _J(_f[l], _u[l]);
    }
    else
    {
        (*_inv)(_f[l], _u[l]);
    }
}
```

The reason

Now we have (no construction of operators):

```
void AMG_J(int k)
{
   int l = k++;
   if (l < _level) {
      // smooth with jacobi algorithm
      for (int i = 0; i < _nsmooth; i++) (*_INV[l])(_f[l], _u[l]);

      // Calculate residual vector _r = _f - _A * _u
      (*_A_solve[l]).Residual( _f[l], _u[l], _r[l]);
      (*_R_solve[l]).RestrictAverage(_r[l], _f[l + 1]); // Restriction operation

      AMG_J(k); // Got to next AMG level

      (*_R_solve[l]).ProlongateAverage(_u[l + 1], _u[l]); // Prolongation operation

      // smooth with jacobi algorithm
      for (int i = 0; i < _nsmooth; i++) (*_INV[l])(_f[l], _u[l]);
   }
   else {   // Calculate inverse on the coarsed grid
      (*_A_solve[l]).DiagonalVector(_f[l], _u[l]);

      if(_prepare_for_device && _network_size > 1) _u[l].updatehost_shared_nodes(
           _nshared[l]);
      _com[l].accumulate(_u[l]);
      if(_prepare_for_device && _network_size > 1) _u[l].updatedev_shared_nodes(_nshared
           [l]);
   }
}
```

PGI-Bugreport Nr.: FS24002

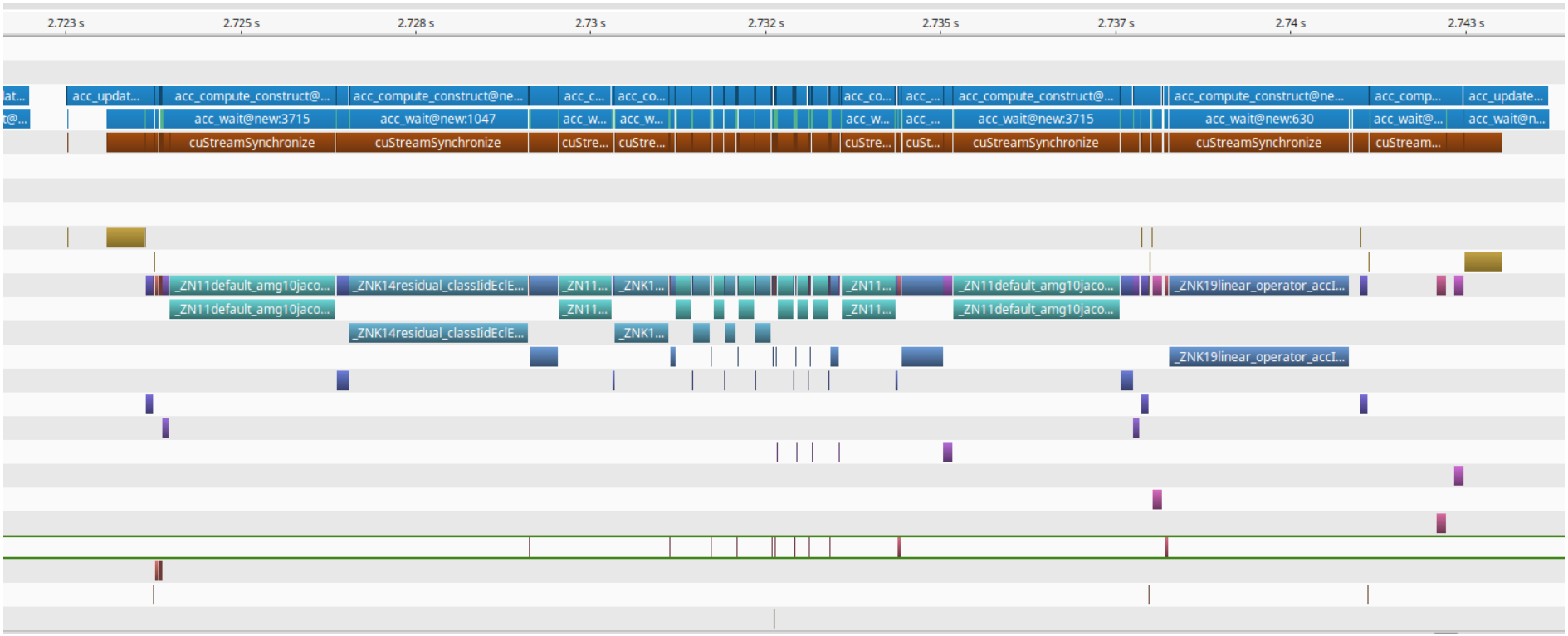


Figure: Now we have this profiling.

Note: Define device **Data and Operators** for OpenACC as early as possible!

PGI-Bugreport Nr.: FS24002

- Why does this behaviour occur?
- OpenACC for C++ → *data regions*

```
int main(int argc, char **argv)
{
    std::vector<double> a(100);
    // deep copy
   double* data = a.data();
   int size = a.size();
  #pragma acc enter data pcopyin(data[0:size])

    {
        function_with_openacc_kernels(data);
        {
            another_function_with_openacc_kernels(data);
        }
    }
    #pragma acc exit data delete(data[0:size])
    return 0;
}
```

PGI-Bugreport Nr.: FS24002

- Why does this behaviour occur?
- OpenACC for C++ → *data regions*

```
int main(int argc, char **argv)
{
   std::vector<double> a(100);
   // deep copy
  const double* data = a.data();
  const int size = a.size();
  #pragma acc enter data pcopyin(data[0:size])

// during compiling, OpenACC creats data regions, for every called constructor!
    --> pragma acc data present(data)
   {
      function_with_openacc_kernels(data);
       --> pragma acc data present(data)
      {
         another_function_with_openacc_kernels(data);
      }
   }
   #pragma acc exit data delete(data[0:size])
   return 0;
}
```

PGI-Bugreport Nr.: FS24002

- Why does this behaviour occur?
- OpenACC for C++ → *data regions*

```
int main(int argc, char **argv)
{
   std::vector<double> a(100);
   // deep copy
  const double* data = a.data();
  const int size = a.size();
  #pragma acc enter data pcopyin(data[0:size])

// during compiling, OpenACC creats data regions, for every called constructor!
    --> pragma acc data present(data)
   {
      function_with_openacc_kernels(data);
       --> pragma acc data present(data)
      {
         another_function_with_openacc_kernels(data);
      }
   }
   #pragma acc exit data delete(data[0:size])
   return 0;
}
```

- OpenACC creates *data regions* for the user!
- For every *data region*, OpenACC has to ask for existing data!

Unified Memory

- OpenACC uses *unified memory* for single values (int, double).
- One has to be careful with *bigger programs*:

```
// _max_length is private member of the class
const int max_length = _max_length;
#pragma acc parallel loop pcopyin(...)
for(int ii = 0; ii < v_size; ii += _block_size) {
   const int stride = ii*max_length;
   const int kk_max = (...);
#pragma acc loop independent
   for(int kk = ii; kk < kk_max; ++kk) {
      S s = 0.0;
#pragma acc loop seq
      for(int jj = 0; jj < max_length; ++jj) {
         // do some parallel stuff
      }
      v[kk] = s;
}}
```

1 Introduction

2 Redesign

3 How to use OpenACC

4 Coalescing for OpenACC

5 Schön

ELLPACK

Example Matrix:

$$A = \begin{pmatrix} 1 & 3 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 8 & 2 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 9 & 0 & 0 & 0 & 7 \end{pmatrix} \rightsquigarrow A = \begin{pmatrix} 1 & 3 & 0 \\ 2 & 0 & 0 \\ 1 & 0 & 0 \\ 3 & 8 & 2 \\ 4 & 0 & 0 \\ 9 & 7 & 0 \end{pmatrix}; col := \begin{pmatrix} 1 & 2 & 1 \\ 2 & 1 & 1 \\ 3 & 1 & 1 \\ 2 & 4 & 5 \\ 5 & 1 & 1 \\ 2 & 6 & 1 \end{pmatrix};$$

ELLPACK

Example Matrix:

$$A = \begin{pmatrix} 1 & 3 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 8 & 2 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 9 & 0 & 0 & 0 & 7 \end{pmatrix} \rightsquigarrow A = \begin{pmatrix} 1 & 3 & 0 \\ 2 & 0 & 0 \\ 1 & 0 & 0 \\ 3 & 8 & 2 \\ 4 & 0 & 0 \\ 9 & 7 & 0 \end{pmatrix}; col := \begin{pmatrix} 1 & 2 & 1 \\ 2 & 1 & 1 \\ 3 & 1 & 1 \\ 2 & 4 & 5 \\ 5 & 1 & 1 \\ 2 & 6 & 1 \end{pmatrix};$$

ELLPACK

Example Matrix:

$$A = \begin{pmatrix} 1 & 3 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 8 & 2 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 9 & 0 & 0 & 0 & 7 \end{pmatrix} \rightsquigarrow A = \begin{pmatrix} 1 & 3 & 0 \\ 2 & 0 & 0 \\ 1 & 0 & 0 \\ 3 & 8 & 2 \\ 4 & 0 & 0 \\ 9 & 7 & 0 \end{pmatrix}; col := \begin{pmatrix} 1 & 2 & 1 \\ 2 & 1 & 1 \\ 3 & 1 & 1 \\ 2 & 4 & 5 \\ 5 & 1 & 1 \\ 2 & 6 & 1 \end{pmatrix};$$

- OpenACC creates GPU Code!
- But one must take care on data access!

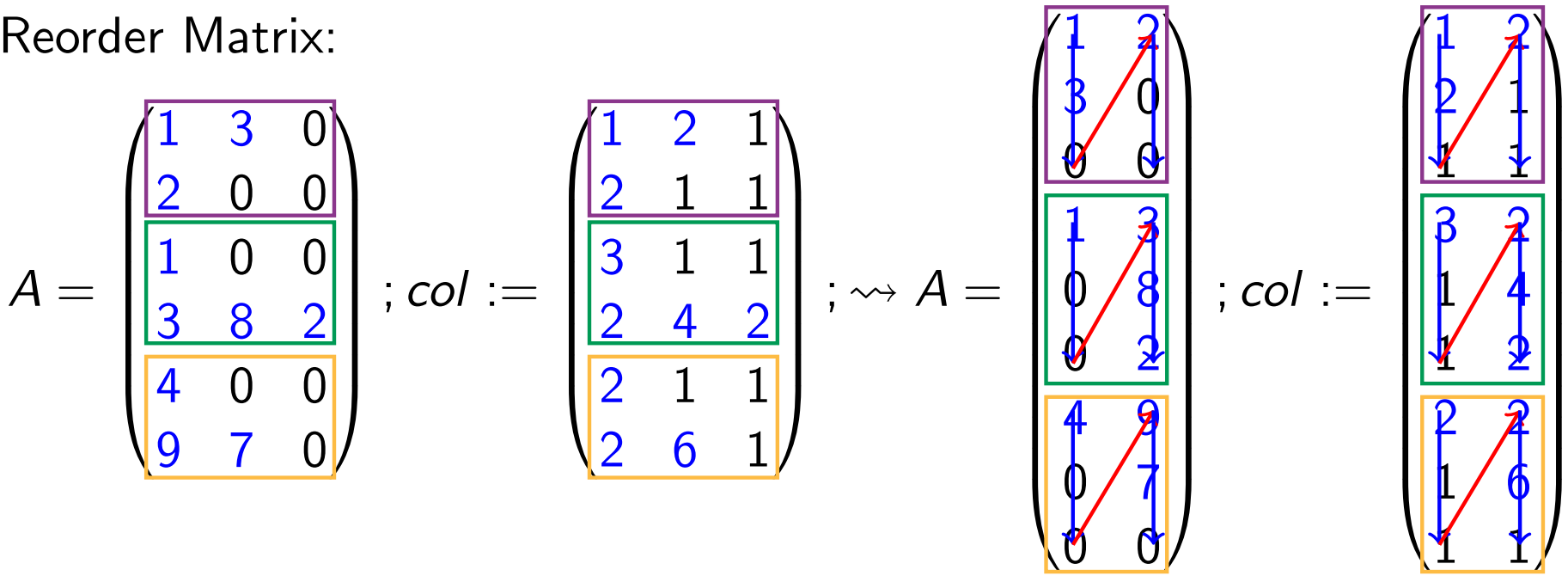
Ellpack

Reorder Matrix:

$$A = \begin{pmatrix} 1 & 3 & 0 \\ 2 & 0 & 0 \\ 1 & 0 & 0 \\ 3 & 8 & 2 \\ 4 & 0 & 0 \\ 9 & 7 & 0 \end{pmatrix}; col := \begin{pmatrix} 1 & 2 & 1 \\ 2 & 1 & 1 \\ 3 & 1 & 1 \\ 2 & 4 & 5 \\ 5 & 1 & 1 \\ 2 & 6 & 1 \end{pmatrix}; \rightsquigarrow A = \begin{pmatrix} 1 & 2 \\ 3 & 0 \\ 0 & 0 \\ 1 & 3 \\ 0 & 8 \\ 0 & 2 \\ 4 & 9 \\ 0 & 7 \\ 0 & 0 \end{pmatrix}; col := \begin{pmatrix} 1 & 2 \\ 2 & 1 \\ 1 & 1 \\ 3 & 2 \\ 1 & 4 \\ 1 & 5 \\ 5 & 2 \\ 1 & 6 \\ 1 & 1 \end{pmatrix}$$

Ellpack

Reorder Matrix:



Coalescing

- It tooked me 2-3 months to implement the coalescing, with correct parallelization!
- Since we have different type of matrices (degrees of freedom), I implemented another solver type $\rightarrow$ another month!

Coalescing

- It tooked me 2-3 months to implement the coalescing, with correct parallelization!
- Since we have different type of matrices (degrees of freedom), I implemented another solver type $\rightarrow$ another month!

- After redesigning of the code, I implemented

20 different types of matrices

in 2 Weeks!!!!!!!

Coalescing

- It tooked me 2-3 months to implement the coalescing, with correct parallelization!
- Since we have different type of matrices (degrees of freedom), I implemented another solver type $\rightarrow$ another month!

- After redesigning of the code, I implemented

20 different types of matrices

in 2 Weeks!!!!!!!

- **Without any discussion: writing nice code is MANDATORY!**

OpenACC the first

We calculate the solution a TBunnyC size mesh on the device:

OpenACC performance

	1 CPU	OpenACC	+ELLPACK	+RELLPACK
Jacobi	9,927	1,808	1,5163	1,1296
impro:		× 5,49	× 6,55	× 8,79
AMG				
impro:				

OpenACC the first

We calculate the solution a TBunnyC size mesh on the device:

OpenACC performance

	1 CPU	OpenACC	+ELLPACK	+RELLPACK
Jacobi	9,927	1,808	1,5163	1,1296
impro:		× 5,49	× 6,55	× 8,79
AMG	2.561	0.5567	0.4491	0.3310
impro:		× 4,60	× 5,70	× 7,74

OpenACC the second

We calculate the solution a 4 times - TBunnyC -size mesh on the device:

OpenACC performance 2x2 Jacobi		
	Solve time	improvement
1 CPU	27,3670	× 1,00
OpenACC:	6,0160	× 4,55
Crowded ELLPACK:	6,2348	× 4,39
Crowded RELLPACK:	4,3782	× 6,25
ELLPACK:	5,4020	× 5,07
RELLPACK:	3,5381	× 7,73
Full RELLPACK:	3,4058	× 8,04
Full RELLPACK+:	3,6728	× 7,45

OpenACC the third

We calculate the solution a 4 times - TBunnyC -size mesh on the device:

OpenACC performance 2x2 AMG

	Solve time	improvement
1 CPU	17,3471	× 1,00
OpenACC:	4,4420	× 3,91
Crowded ELLPACK:	5,0751	× 3,42
Crowded RELLPACK:	3,3190	× 5,23
ELLPACK:	4,2340	× 4,10
RELLPACK:	2,8301	× 6,13
Full RELLPACK:	2,7081	× 6,41
Full RELLPACK+:	2,9454	× 5,89

OpenACC the forth

We calculate the solution a 9 times - TBunnyC -size mesh on the device:

OpenACC performance 3x3 Jacobi		
	Solve time	improvement
1 CPU	55,6129	× 1,00
OpenACC:	17,0170	× 3,27
Crowded ELLPACK:	14,5486	× 3,82
Crowded RELLPACK:	9,3075	× 5,98
ELLPACK:	11,6697	× 4,77
RELLPACK:	8,3764	× 6,64
Full RELLPACK:	7,3215	× 7,60
Full RELLPACK+:	7,5949	× 7,32
Full RELLPACK-DV:	7,4230	× 7,49

OpenACC the forth

We calculate the solution a modified - TBunnyC - size mesh on the device:

OpenACC performance 3x3 AMG		
	Solve time	improvement
1 CPU	9,5729	× 1,00
OpenACC:	3,2889	× 2,91
Crowded ELLPACK:	3,7271	× 2,57
Crowded RELLPACK:	2,0427	× 4,69
ELLPACK:	2,9771	× 3,21
RELLPACK:	1,9077	× 4,84
Full RELLPACK:	1,7076	× 5,61
Full RELLPACK+:	1,7722	× 5,40
Full RELLPACK-DV:	1,7448	× 5,49

1 Introduction

2 Redesign

3 How to use OpenACC

4 Coalescing for OpenACC

5 Schön

Jan Hendrik Schön

- Received the Otto-Klung-Weberbank Prize for Physics
- Received the Braunschweig Prize in 2001
- Received the Outstanding Young Investigator Award of the Materials Research Society in 2002

Jan Hendrik Schön

- Received the Otto-Klung-Weberbank Prize for Physics
- Received the Braunschweig Prize in 2001
- Received the Outstanding Young Investigator Award of the Materials Research Society in 2002

- In 2001 he was listed as an author on an average of one newly published research paper every eight days!

Jan Hendrik Schön

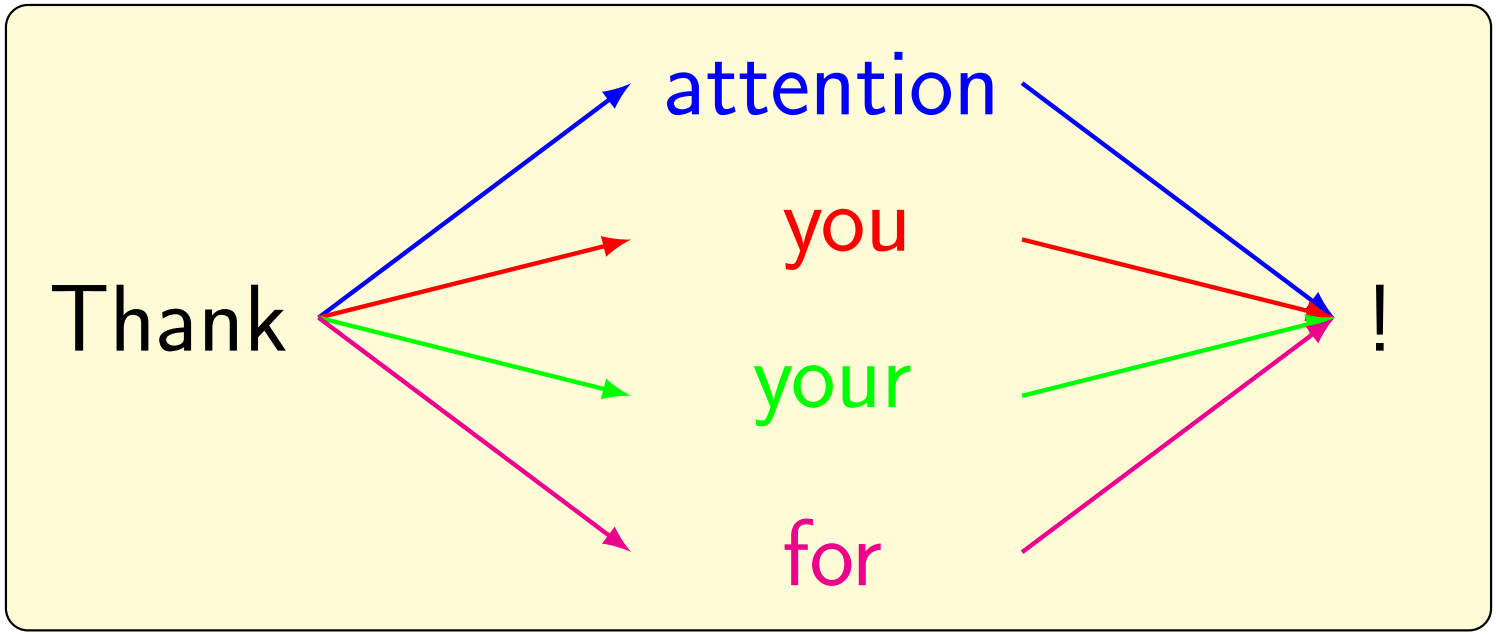
- Received the Otto-Klung-Weberbank Prize for Physics
- Received the Braunschweig Prize in 2001
- Received the Outstanding Young Investigator Award of the Materials Research Society in 2002

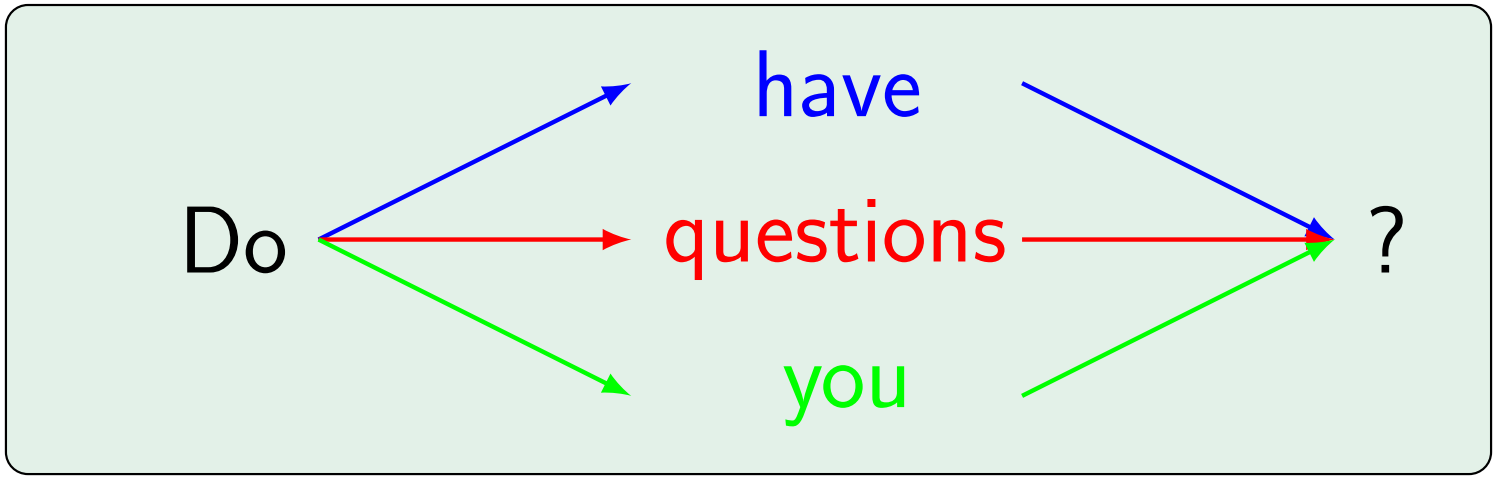
- In 2001 he was listed as an author on an average of one newly published research paper every eight days!
- September 25, 2002: A committee report published contained details of 24 allegations of misconduct! $\longrightarrow$ even his colleagues had no chance to double check his *work*!

Jan Hendrik Schön

- Received the Otto-Klung-Weberbank Prize for Physics
- Received the Braunschweig Prize in 2001
- Received the Outstanding Young Investigator Award of the Materials Research Society in 2002

- In 2001 he was listed as an author on an average of one newly published research paper every eight days!
- September 25, 2002: A committee report published contained details of 24 allegations of misconduct! $\longrightarrow$ even his colleagues had no chance to double check his *work*!
- Conclusion for us: **A source code without documentation, can not be considered as science!!!!!!**
  c.f.: `https://en.wikipedia.org/w/index.php?title=Sch%C3%B6n_scandal&oldid=785900520`


Thank
attention
you
your
for
!


Do
have
questions
you
?