

Performance Tools for Technical Computing

Christian Terboven

terboven@rz.rwth-aachen.de

Center for Computing and Communication

RWTH Aachen University

Agenda

- Motivation and Methodology
- Profiling Tools
- Parallel Profiling Tools
- Debugging and Correctness Tools
- Summary

2

Center for

Computing and

Communication

Motivation

Profiling

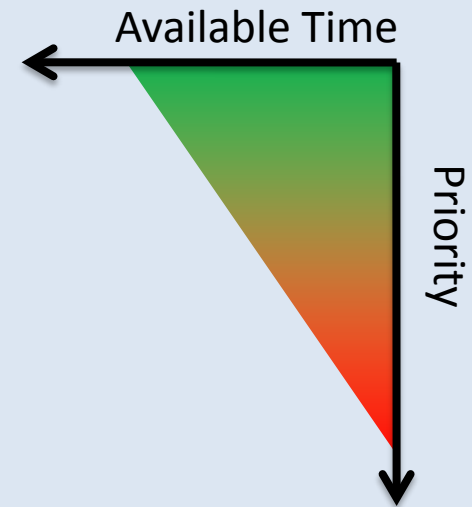
Par. Profiling

Debugging

Summary

My Starting Point (1/2)

- HPC for Engineers and Scientists:
 - Actual (Engineering / Research) work
 - Familiarizing with the code
 - Debugging
 - Parallelization
 - Scheduling
 - Tuning



→ Productivity becomes more and more of an issue!

- What to aim for?
 - Trivial Scalability? → Parametric Array Jobs
 - Parallel Scalability? → Parallelization with MPI, OpenMP, ...
 - Architecture-specific tuning? → cc-NUMA, Cache Access, ...

3

enter for

computing and

communication

Motivation

Profiling

Par. Profiling

Debugging

Summary

My Starting Point (2/2)

- HPC systems in Aachen:
 - Today: About 700 x86-based systems with two or more sockets, Intel Harpertown + Nehalem-EP, 2-3 GB of memory per core
 - Tomorrow: About 210 TFLOPS with Intel Nehalem-EX in four to eight sockets, still about 2-3 GB of memory per core
 - CentOS Linux (92% of the nodes) and Windows HPC Server (8%)

 - Compared to five years ago the usage scenario has changed
 - January 2010:
 - Over 64k batch jobs, 10% of CPU time for undergraduate students
 - Heterogeneous job-mix: From large parallel to large job array
 - Applications consist of multiple languages, i.e. Matlab + Fortran
- We hardly can force users to follow the *traditional* HPC path

4

Center for

Computing and

Communication

Motivation

Profiling

Par. Profiling

Debugging

Summary

Methodology: Tuning and Parallelization

- (Incremental) Tuning and Parallelization process:
 - (1) Runtime analysis of a given program
 - (2) Identification of so-called *Hotspots*, that are compute intensive parts of a program → Think about serial tuning
 - (1) If serial tuning was applicable → Restart at (1)
 - (3) Parallelization of identified *Hotspots*
 - (4) Restart at (1), but switch to parallel runtime analysis
- Important tools in this process:
 - Profiler for serial and parallel program
 - Debugger
 - If applicable: tools for correctness checking

5

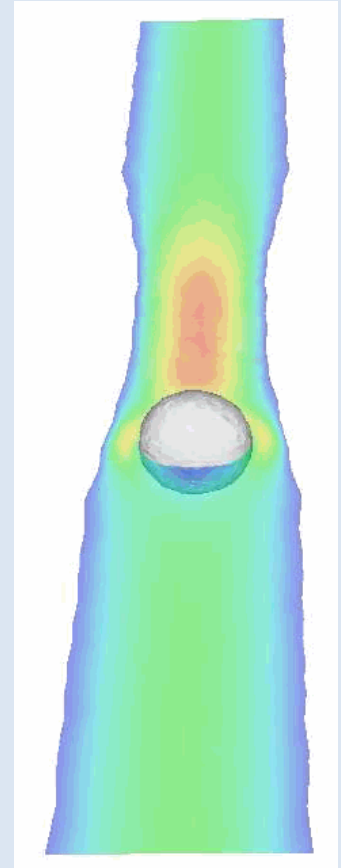
Application used in this Case Study: DROPS

- Numerical Simulation of two-phase flow
- (Adaptive) Tetrahedral Grid Hierarchy
- Finite Element Method (FEM)

- Written in C++: is object-oriented, uses nested templates, uses STL types, uses compile-time polymorphism, ...

- Typical Setup:
 - Visual Studio *Release* Configuration, 64bit
 - Tiny dataset: 77.5 seconds per run
 - Benchmark kernels for identified Hotspots

Example:
Silicon oil drop in
D₂O (fluid/fluid)



6

enter for

Computing and

Communication

Motivation

Profiling

Par. Profiling

Debugging

Summary

Agenda

- Motivation and Methodology
- Profiling Tools
- Parallel Profiling Tools
- Debugging and Correctness Tools
- Summary

7

Center for

Computing and

Communication

Motivation

Profiling

Par. Profiling

Debugging

Summary

Profiler: Requirements

- Goals of this analysis step:
 - Get an overview of the program's call tree
 - Get an overview of how much compute time is spent in which parts of the program
 - Inclusive time per function
 - Exclusive time per function
 - Derive program's critical path with respect to performance

- Comparison of the following tools:
 - Visual Studio 2008 / gprof
 - Intel Parallel Studio: Amplifier
 - Intel VTune

8

enter for

Computing and

Communication

Motivation

Profiling

Par. Profiling

Debugging

Summary

Visual Studio 2008 / gprof (1/2)

- Execution time: 81.7 seconds over 77.5 seconds wo/ tool
- Functions: Inclusive time versus Exclusive time, per function

Current View: Functions

Function Name	Inclusive Samples	Exclusive Sam...
DROPS::y_Ax<double>(double *,unsigned __int64,double const *,unsigned __int64 const *,un	8.398	8.398
DROPS::y_ATx<double>(double *,unsigned __int64,double const *,unsigned __int64 const *,un	5.950	5.950
[ntdll.dll]	22.716	1.642
std::valarray<double>::_Grow(unsigned __int64,double const *,unsigned __int64)	2.084	1.312
DROPS::operator* <class DROPS::SparseMatBaseCL<double>,class DROPS::SparseMatBaseCL<	12.773	474
DROPS::InstatNavierStokes2PhaseP2P1CL<class DROPS::ZeroFlowCL>::SetupNonlinear_P2(cla	2.509	393
DROPS::dot(class DROPS::GridFunctionCL<class DROPS::SVectorCL<3> > const &,class DROPS	759	383
DROPS::SparseMatBuilderCL<double>::operator()(unsigned __int64,unsigned __int64)	470	246
std::operator* <double>(double const &,class std::valarray<double> const &)	370	245
DROPS::SparseMatBuilderCL<double>::Build(void)	615	230
[MSVCR90.dll]	1.886	222
std::_Tree<class std::_Tmap_traits<unsigned __int64,class DROPS::FaceCL *,struct std::less<	319	186
std::_Allocate<unsigned __int64>(unsigned __int64,unsigned __int64 *)	184	184
DROPS::ExchangeCL::AccurLocDotNoAcc_(class DROPS::VectorBaseCL<double> const &,class I	175	175
DROPS::ParModGMRES<class DROPS::MLSparseMatBaseCL<double>,class DROPS::VectorBase	3.485	157
DROPS::Quad5CL<double>::quadP2(int,double)const	213	154
DROPS::LevelsetP2CL::SetupSystem<class DROPS::P2EvalCL<class DROPS::SVectorCL<3>,cla	904	147
std::valarray<double>::operator=(class std::valarray<double> const &)	141	141
DROPS::FE_P2CL::val<class DROPS::LocalP2CL<double> >(class DROPS::LocalP2CL<double>	13	
std::_Tree<class std::_Tmap_traits<unsigned __int64,class DROPS::FaceCL *,struct std::less<	45	
DROPS::ParAccurPCG<class DROPS::CompositeMatrixBaseCL<class DROPS::SparseMatBaseCL<	13.13	
DROPS::SetupSystem1_P2<class DROPS::ZeroFlowCL>(class DROPS::MultiGridCL const &,class	1.29	
DROPS::SetupSystem2_P2P1X<class DROPS::ZeroFlowCL>(class DROPS::MultiGridCL const &,cl	32	

→ Provides a good overview of where the time is spent.

Visual Studio 2008 / gprof (2/2)

o Caller / Callee: Call tree browsing, metrics per call site

Current View: Caller / Callee

Functions that called DROPS::Strategy<class DR

Function Name	Inclusive Samples	Exclusive Samples
main	22.558	0

Current function

DROPS::Strategy<class DROPS::ZeroF	22.558	0
------------------------------------	--------	---

Functions that were called by DROPS::Strategy<

DROPS::AdapTriangCL::UpdateTriang(211	0
DROPS::CreateTimeDisc<class DROPS:	856	0
DROPS::DisplayDetailedGeom(class DR	1	0
DROPS::DisplayUnks<class DROPS::In	1	0
DROPS::IdxDescCL::CreateNumbering	3	0

→ Allows for very good examination of the (dynamic) program structure and making tuning and / or parallelization decisions.

- Missing:
- Simple identification of Hotspots
 - Good mapping to the source code
 - Support for parallelism

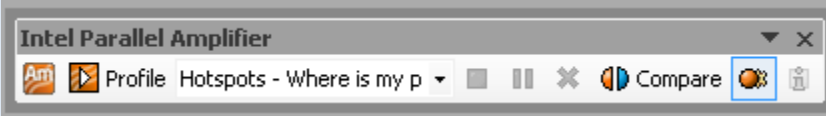
Current View: Call Tree

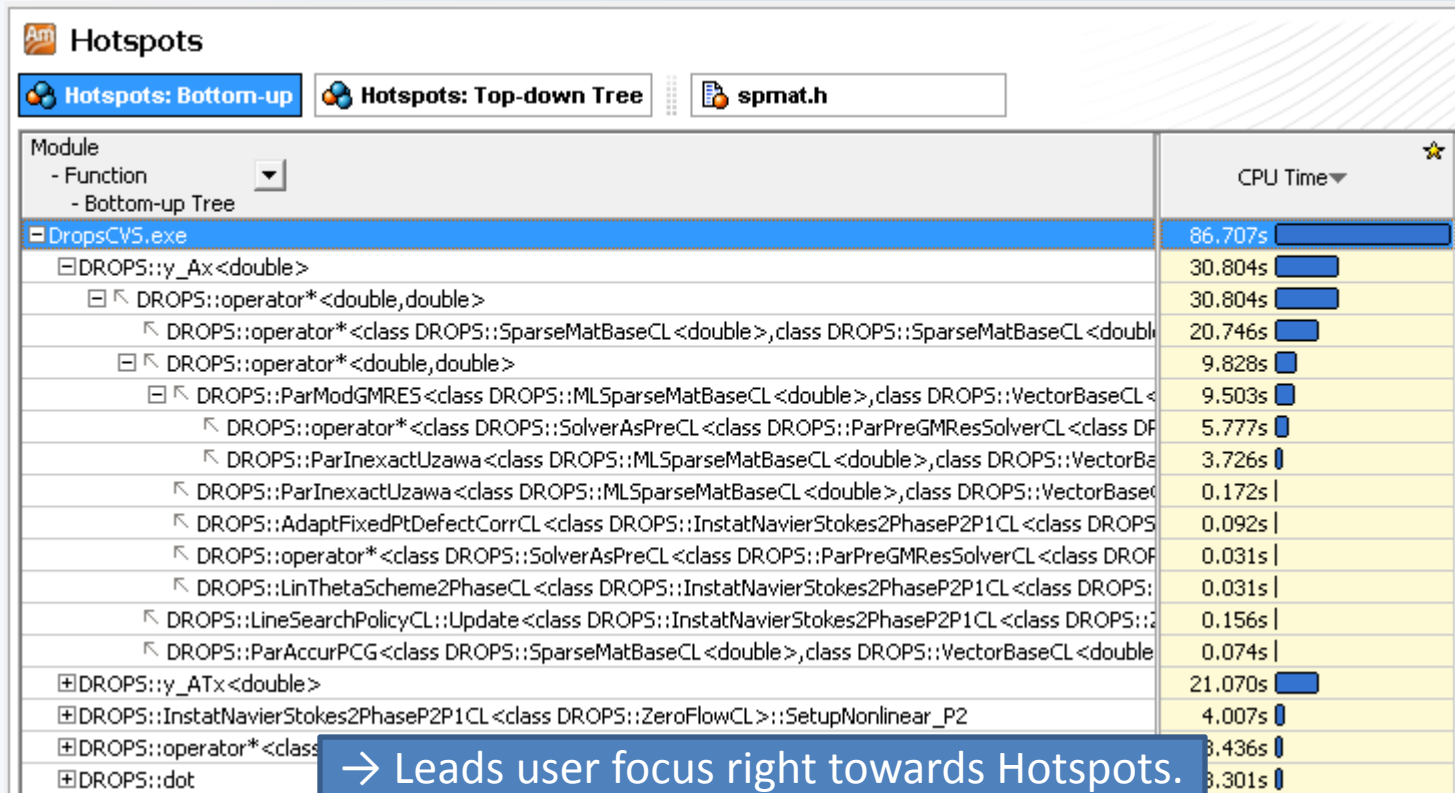
Noise Reduction is enabled for this view. [Configure...](#)

Function Name	Inclusive Sam...	Exclusive Samples
[-] DropsCVS.exe	22.717	0
[-] [5 rows folded ending in "DROPS::Strategy<class DROPS::ZeroFlowCL>(class DROPS::InstatNavierStokes2Ph	22.558	0
[-] [1] DROPS::LinThetaScheme2PhaseCL<class DROPS::InstatNavierStokes2PhaseP2P1CL<class DROPS::ZeroF	847	0
[-] [2 rows folded ending in "DROPS::LinThetaScheme2PhaseCL<class DROPS::InstatNavierStokes2PhaseP2F	856	0
[-] [1] DROPS::LinThetaScheme2PhaseCL<class DROPS::InstatNavierStokes2PhaseP2P1CL<class DROPS::ZeroF	856	0
[-] [2 rows folded ending in "DROPS::LinThetaScheme2PhaseCL<class DROPS::InstatNavierStokes2PhaseP2F	20.430	0
[+] DROPS::InstatStokes2PhaseP2P1CL<class DROPS::ZeroFlowCL>::SetupSystem1(class DROPS::MatDescf	1.295	0
[+] DROPS::AdaptFixedPtDefectCorrCL<class DROPS::InstatNavierStokes2PhaseP2P1CL<class DROPS::Zero	18.818	0

10

Intel Parallel Studio: Amplifier (1/2)

- Execution time: 87.6 seconds over 77.5 seconds wo/ tool
- Available from within VS: 
- Hotspot-based display of analysis result: → Easily accessible.



Module	CPU Time
DropsCVS.exe	86.707s
DROPS::y_Ax<double>	30.804s
DROPS::operator*<double,double>	30.804s
DROPS::operator*<class DROPS::SparseMatBaseCL<double>,class DROPS::SparseMatBaseCL<double>	20.746s
DROPS::operator*<double,double>	9.828s
DROPS::ParModGMRES<class DROPS::MLSparseMatBaseCL<double>,class DROPS::VectorBaseCL<	9.503s
DROPS::operator*<class DROPS::SolverAsPreCL<class DROPS::ParPreGMRESolverCL<class DF	5.777s
DROPS::ParInexactUzawa<class DROPS::MLSparseMatBaseCL<double>,class DROPS::VectorBa	3.726s
DROPS::ParInexactUzawa<class DROPS::MLSparseMatBaseCL<double>,class DROPS::VectorBase	0.172s
DROPS::AdaptFixedPtDefectCorrCL<class DROPS::InstatNavierStokes2PhaseP2P1CL<class DROPS	0.092s
DROPS::operator*<class DROPS::SolverAsPreCL<class DROPS::ParPreGMRESolverCL<class DROF	0.031s
DROPS::LinThetaScheme2PhaseCL<class DROPS::InstatNavierStokes2PhaseP2P1CL<class DROPS:	0.031s
DROPS::LineSearchPolicyCL::Update<class DROPS::InstatNavierStokes2PhaseP2P1CL<class DROPS::	0.156s
DROPS::ParAccurPCG<class DROPS::SparseMatBaseCL<double>,class DROPS::VectorBaseCL<double	0.074s
DROPS::y_ATx<double>	21.070s
DROPS::InstatNavierStokes2PhaseP2P1CL<class DROPS::ZeroFlowCL>::SetupNonlinear_P2	4.007s
DROPS::operator*<class	8.436s
DROPS::dot	8.301s

11

Center for

Computing and
Communication

Motivation

Profiling

Par. Profiling

Debugging

Summary

Intel Parallel Studio: Amplifier (2/2)

- o Examination of analysis result at the level of source lines:

897	// y= A*x	
898	// fails, if num_rows==0.	
899	// Assumes, that none of the arrays involved do alias.	
900	template <typename T>	
901	inline void	
902	y_Ax(T* __restrict y,	
903	size_t num_rows,	
904	const T* __restrict Aval,	
905	const size_t* __restrict Arow,	
906	const size_t* __restrict Acol,	
907	const T* __restrict x)	
908	{	
909	T sum;	
910	size_t rowend;	
911	size_t nz= 0;	
912	do {	
913	rowend= *++Arow;	0.059s
914	sum= T();	0.031s
915	for (; nz<rowend; ++nz)	0.187s
916	sum+= (*Aval++)*x[*Acol++];	30.433s
917	(*y++)= sum;	0.094s
918	} while (--num_rows > 0);	
919	}	

→ Results surprise users very often, thus this view should be examined before the first efforts towards parallelization are carried out.

Missing:

- Hardware Counter measurements
- Better representation of call graph

The big question: When to stop?

- There is no single correct answer for this questions.
- Today's programming languages abstract from the hardware
 - You can just compile and run your program, and be lucky (or not)
 - You can take a closer look:
 - How many floating point operations does your compute kernel perform per CPU cycle? Your Nehalem-EX can do up to four of them per core per cycle!
 - How much memory bandwidth does your compute kernel consume? Your Nehalem-EX delivers about 10 GB/s for a single thread.
 - And from here you can dive even deeper...

13

Center for

Computing and

Communication

Motivation

Profiling

Par. Profiling

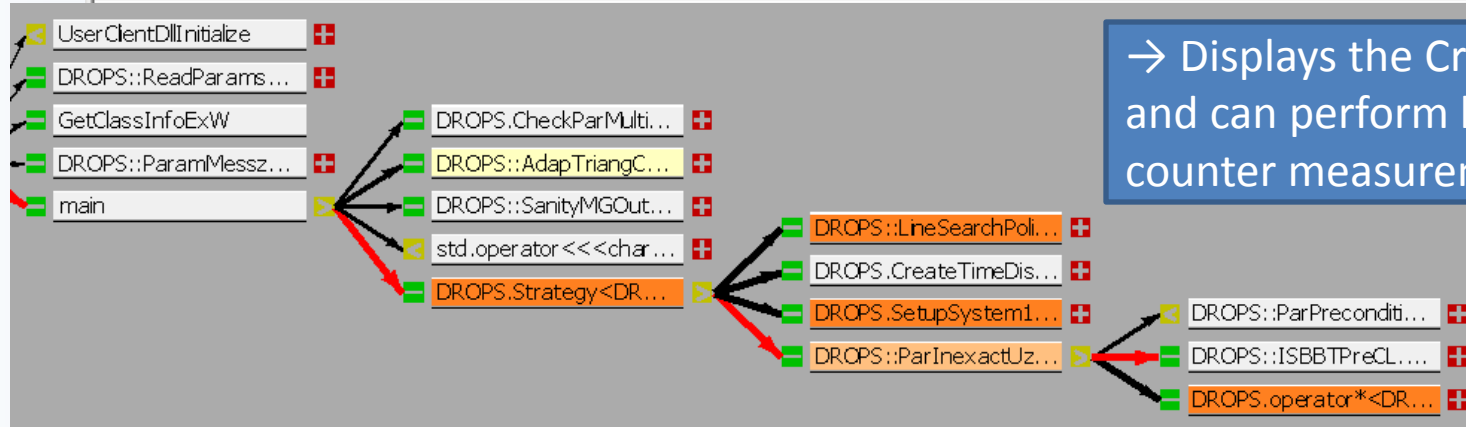
Debugging

Summary

Intel VTune

- o Execution time:
 - Sampling: Failed for this application on our systems
 - Call Graph Profiling: 77.6 Seconds over 77.5 seconds wo/ tool
- o Weak function profile, but call graph view:

Function (55)	Class (55)	Calls (55)	Self Time (55)	Total Time (55)	Callers (55)
LocNorm_sq	DROPS::ExchangeCL	9.327	176.337	177.574	2
main		1	2.092	76.613.213	1
mainCRTStartup		1	0	76.613.811	1
MakeInitialTriang<double __cdecl(DROPS::S...	DROPS::AdapTriangCL	1	159.924	380.580	1
Map	DROPS::AffineSquareCL	2.754	19	19	1
map<unsigned __int64,double,std::less<unsig...	std::map<unsigned __int64,double,std::...	342.387	967	22.242	1
MatDescBaseCL<DROPS::MLSparseMatBas...	DROPS::MatDescBaseCL<DROPS::M...	10	4	20	4



→ Displays the Critical Path, and can perform hardware counter measurements.

Agenda

- Motivation and Methodology
- Profiling Tools
- Parallel Profiling Tools
- Debugging and Correctness Tools
- Summary

15

enter for

Computing and

Communication

Motivation

Profiling

Par. Profiling

Debugging

Summary

Parallel Profiling: Requirements

- Goals of this analysis:
 - Compare how the performance profile has changed to the serial program by the current parallelization
 - Evaluate the scalability and efficiency of the parallelization
 - How much overhead has been introduced?
 - How good is the work distributed to the threads (load balance)?
 - Understand how far you still have to go ...

- Comparison of the following tools:
 - Intel Parallel Studio: Inspector
 - Intel Thread Profiler
 - Intel Vtune

16

enter for

Computing and

Communication

Motivation

Profiling

Par. Profiling

Debugging

Summary

Intel Parallel Studio: Amplifier (1/3)

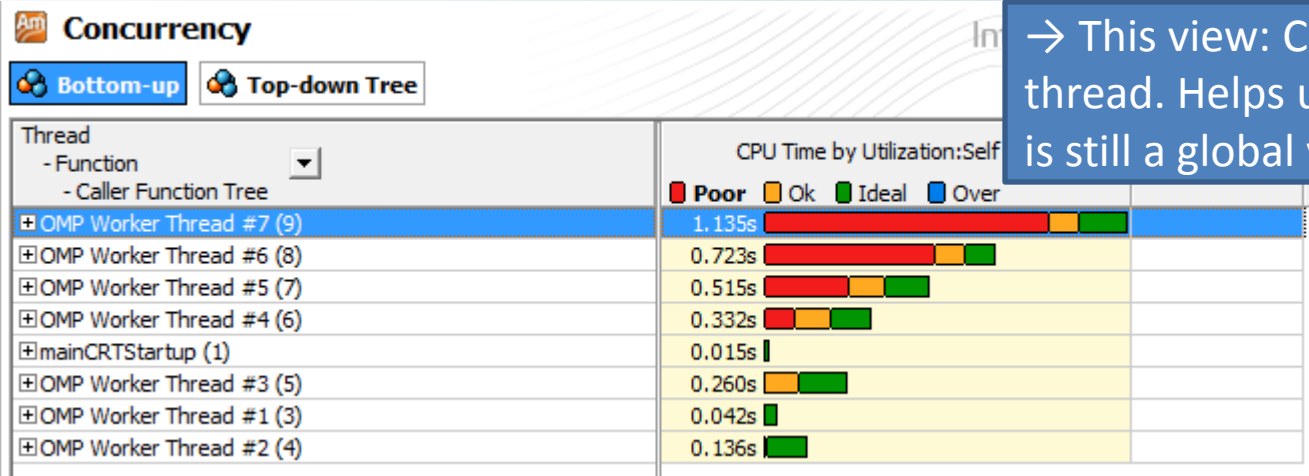
- o Simple thread utilization presentation per function:

The screenshot shows the 'Concurrency: Bottom-up' view in Intel Parallel Studio Amplifier. The table displays CPU time by utilization for various functions. The utilization is categorized into four levels: Poor (red), Ok (orange), Ideal (green), and Over (blue). The function '_drv_initomp1' shows the highest utilization, marked as 'Ideal' (green bar).

Function	Module	CPU Time	Utilization
r_time	smxv-benchmark.exe	0.125s	Poor
y_Ax_omp	smxv-benchmark.exe	0.337s	Poor
main ← _tmainCRTStartup ← BaseThreadInitThunk ← RtlUserThreadStart	smxv-benchmark.exe		
mainomp1	smxv-benchmark.exe		
vcomp::fork_helper ← _vcomp::ParallelRegion::HandlerThreadFunction	smxv-benchmark.exe		
main ← _tmainCRTStartup ← BaseThreadInitThunk ← RtlUserThreadStart	smxv-benchmark.exe		
y_Ax_serial	smxv-benchmark.exe	0.077s	Poor
drv_init	smxv-benchmark.exe	0.051s	Poor
main	smxv-benchmark.exe	0.031s	Poor
drv_initomp1	smxv-benchmark.exe	1.425s	Ideal
_CxxSetUnhandledExceptionFilter	smxv-benchmark.exe	0.000s	

→ First view: basic information.

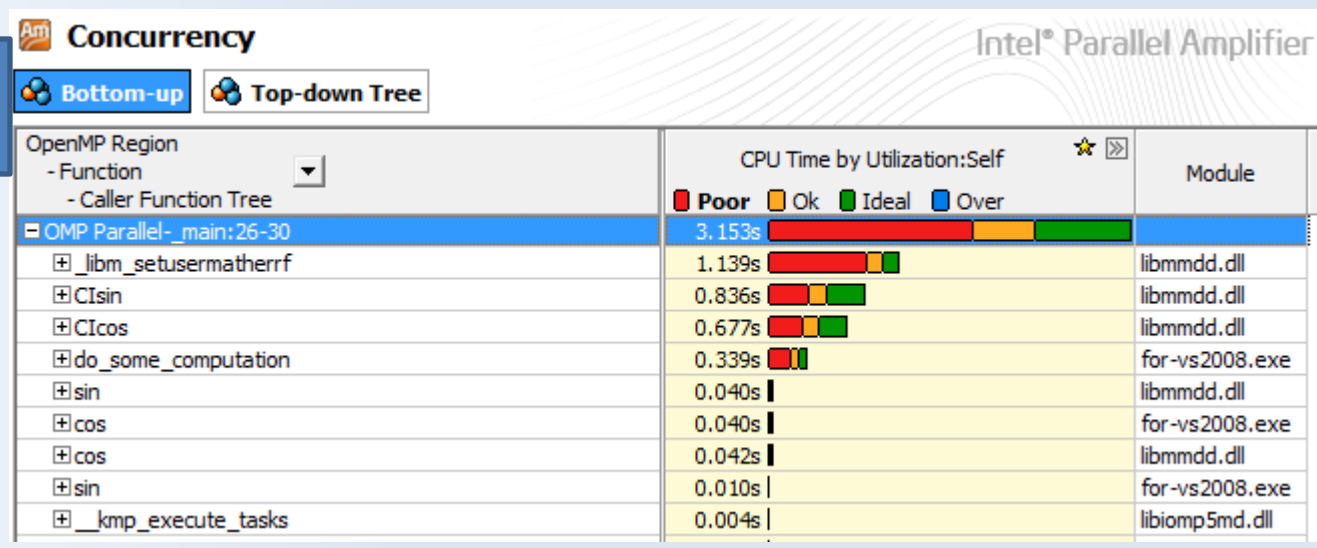
Intel Parallel Studio: Amplifier (2/3)



→ This view: CPU utilization by thread. Helps us further, but it is still a global view.

→ This view: CPU utilization by OpenMP region.

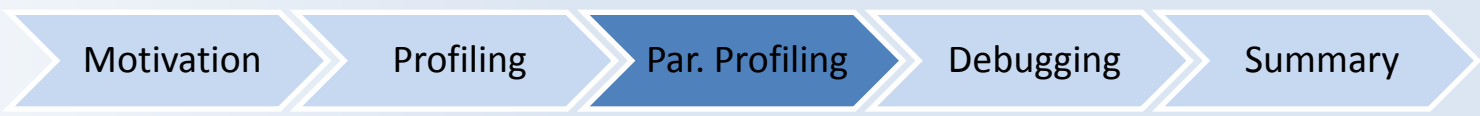
→ Performance tools should be aware of the paradigm!



18

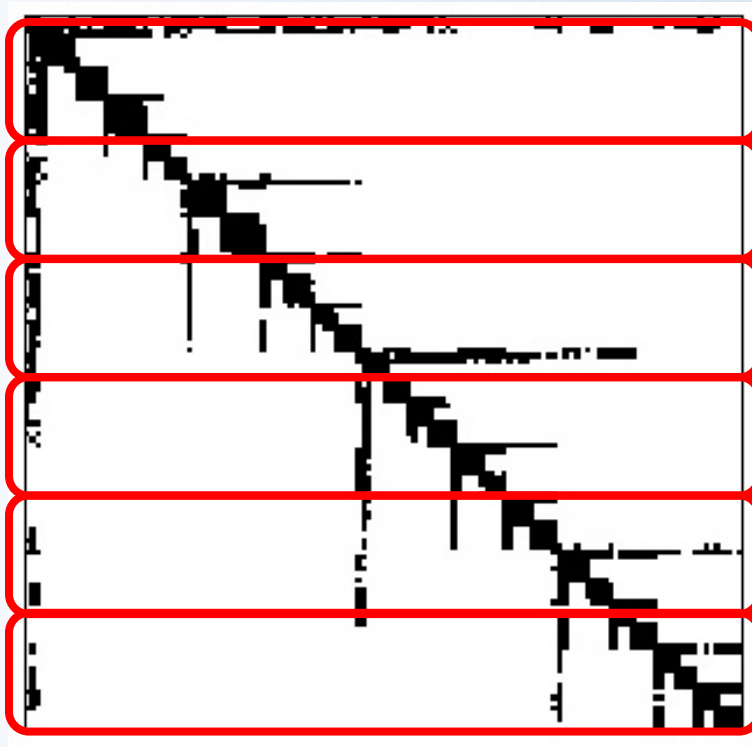
Enter for

Computing and Communication



Intel Parallel Studio: Amplifier (3/3)

- The explanation: If you parallelize a sparse matrix-vector multiplication in the naive way, that is by distributing the rows to threads, you may end up in a load balance due to the structure of the matrix.



Missing:

- Expert information, i.e. the ability to add hardware counter measurements

→ Different views for non OpenMP programs are available as well.

19

enter for

Computing and
Communication

Motivation

Profiling

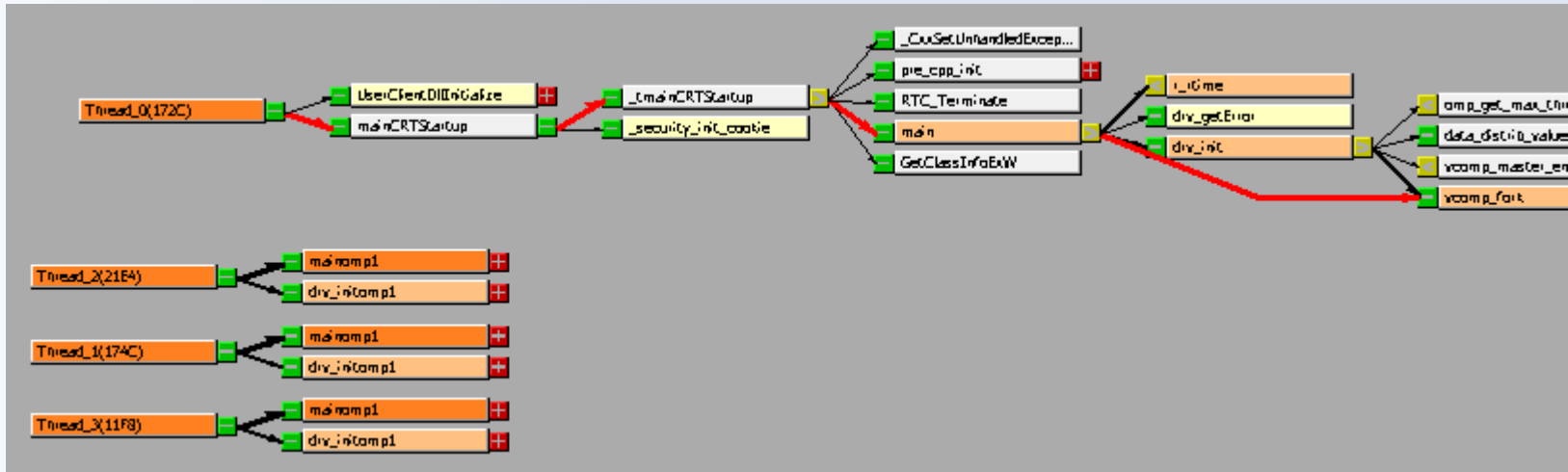
Par. Profiling

Debugging

Summary

Intel Thread VTune

- Call graph profile for multiple threads:



→ In most cases of only little use for *parallel algorithms*.

→ For example: Hardware counter measurements should be put into a context – I am not interested in the sheer number of cache misses, but whether my program is doing good or bad, i.e. how much memory bw my program is consuming per thread.

20

Center for

Computing and
Communication

Motivation

Profiling

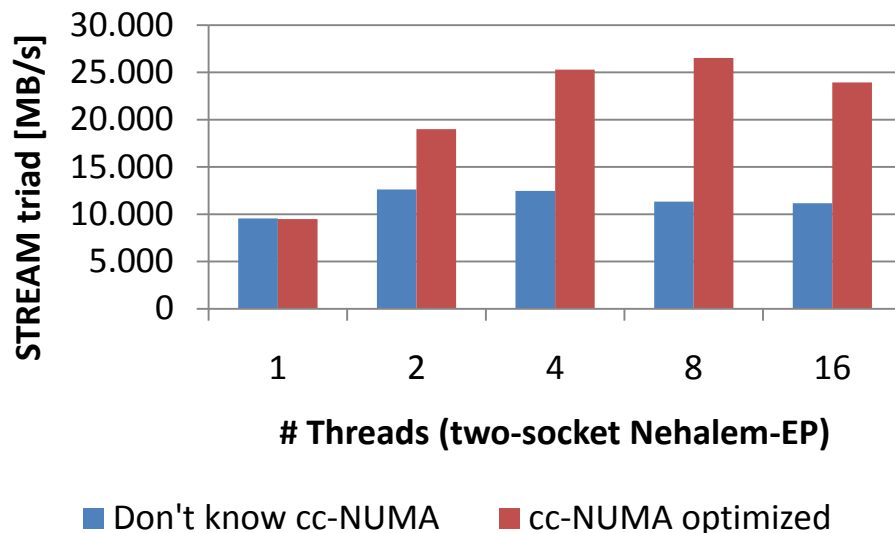
Par. Profiling

Debugging

Summary

Again: How far to you want to go?

- There is still no single correct answer for this questions.
- Today's parallelization paradigms abstract from the hardware
 - You can just compile and run your program, and be lucky (or not)
 - You can take a closer look:
 - Is your program aware of the memory hierarchy? That means does it care for data-to-thread affinity on cc-NUMA architectures?
 - Could it be that two (or more) threads are writing to the same cache line. thus causing False Sharing?



It is often said that Shared-Memory parallelization is simpler than Message-Passing, but it wouldn't scale. That is not true, there is just more to it than loop-parallelization...

Agenda

- Motivation and Methodology
- Profiling Tools
- Parallel Profiling Tools
- Debugging and Correctness Tools
- Summary

22

enter for

Computing and

Communication

Motivation

Profiling

Par. Profiling

Debugging

Summary

Parallel Debugging and Checking: Requirements

- Goals of this analysis:
 - Traditional bug hunting, but with parallel programs
 - Verification that no errors have been introduced by parallelization
 - Check for serial equivalence?
 - Find and eliminate any kind of bug in a parallel program

- Comparison of the following tools:
 - Visual Studio 2008 Debugger
 - Visual Studio 2008 Debugger with Allinea DDTlite
 - Intel Parallel Studio: Inspector (read: Intel Thread Checker)
 - Intel Parallel Studio: Composer

23

enter for

Computing and

Communication

Motivation

Profiling

Par. Profiling

Debugging

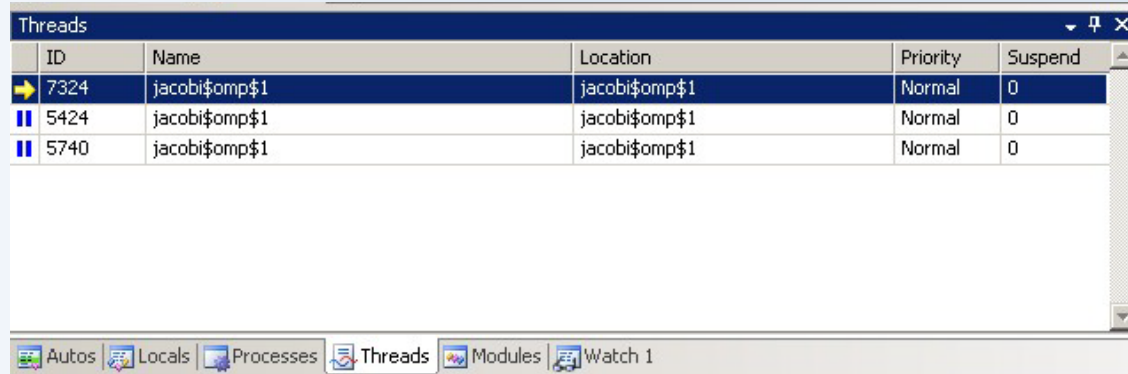
Summary

Visual Studio (2008) Debugger

- Visual Studio 2010 brings well-known debugging experience to multi-threaded programs:

```
65:   k = 1;
66:
67:   while (k <= maxit && error > tol) {
68:
69:       error = 0.0;
70:   #pragma omp parallel private (i)
71:   {
72:   #pragma omp for
```

- Individual control of threads:



→ Very good C++ debugger with all required functions for multi-threaded debugging.

Missing:

- Better control of thread or process groups
- Laminated view of private variables

24

Enter for

Computing and
Communication

Motivation

Profiling

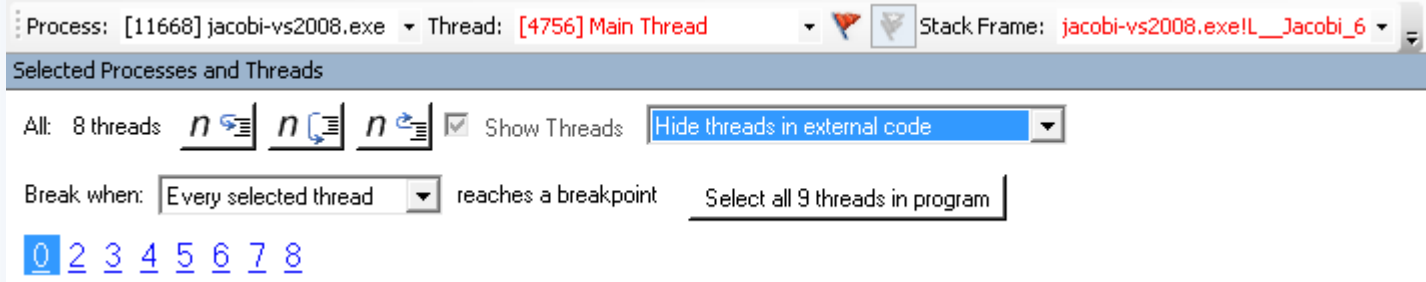
Par. Profiling

Debugging

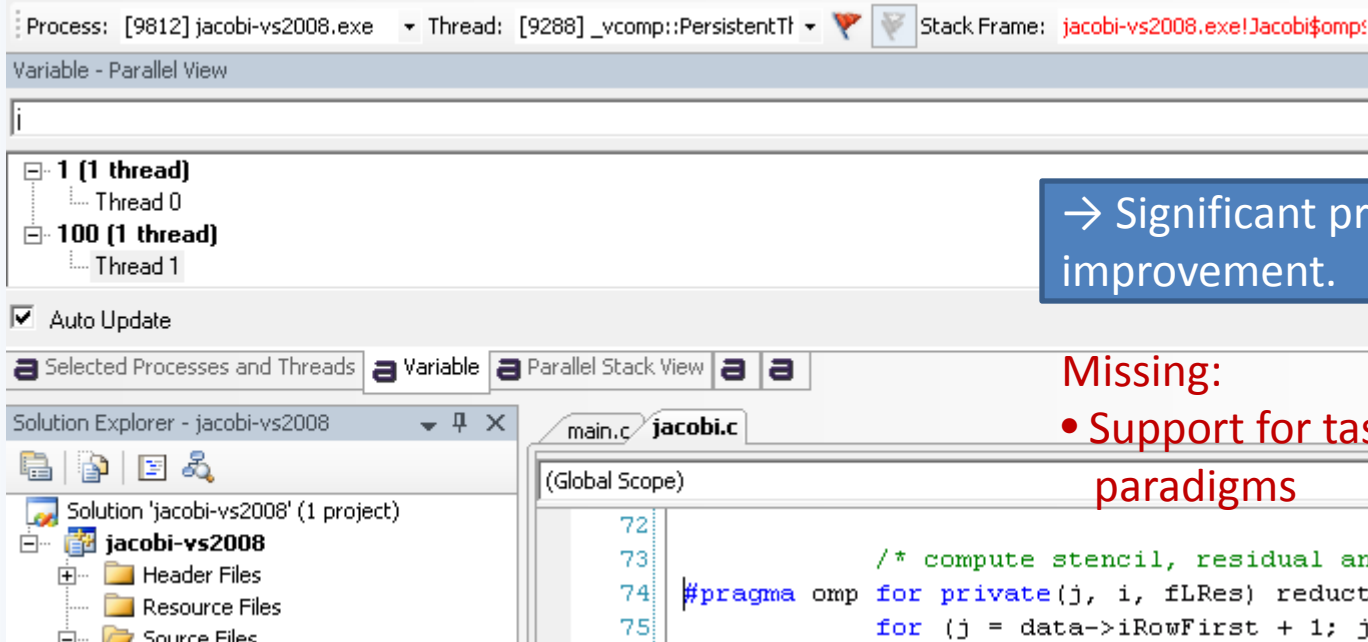
Summary

Visual Studio 2008 Debugger w/ DDTlite

- o Individual thread grouping and switching:



- o Display of variable values per thread:



→ Significant productivity improvement.

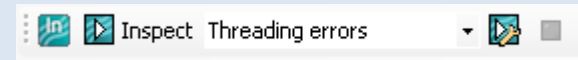
Missing:
• Support for task-based paradigms

25

enter for

Intel Parallel Studio: Inspector

- **Data Race:** the typical OpenMP programming error, when:
 - two or more threads of a *single process* access the same memory location concurrently (between two synchronization points), and at least one of these accesses modifies this location, and the accesses to this location are not protected by locks or critical regions.



→ Easily accessible.

- Non-deterministic occurrence

- (OpenMP-specific) automated data race detection:

Relation Sets	ID /	Short Description	Severity	Description	Count	
1	1	Read -> Write data-race		Memory write at "main.c":196 conflicts with a prior memory read at "main.c":125 (anti...	1	False
2	2	Write -> Read data-race		Memory read at "jacobi.c":61 conflicts with a prior memory write at "jacobi.c":53 (flow dependence)	70	False
2	3	Write -> Read data-race		Memory read at "jacobi.c":61 conflicts with a prior memory write at "jacobi.c":52 (flow dependence)	70	False
2	4	Write -> Read data-race		Memory read at "jacobi.c":61 conflicts with a prior memory write at "jacobi.c":51 (flow...	70	False
2	5	Write -> Read data-race		Memory read at "jacobi.c":61 conflicts with a prior memory write at "main.c":196 (flow...	70	False
2	6	Write -> Write data-race		Memory write at "jacobi.c":66 conflicts with a prior memory write at "jacobi.c":69 (outp...	70	False
2	7	Read -> Write		Memory write at "jacobi.c":66 conflicts with a prior...	59	False

→ Our recommendation: Never put an OpenMP program in production without using this tool.

26

Center for

Computing and
Communication

Motivation

Profiling

Par. Profiling

Debugging

Summary

Intel Parallel Studio: Composer

- Provides new debugging views, especially for tasks:

```
81 static void quick_sort( int lt, int rt, float *data )
82 {
83     if ( (rt-lt) < LOW_LIMIT ) {
84         serial_quick_sort( lt, rt, data );
85     }
86     else {
87         int md = partition( lt, rt, data );
88         #pragma omp task
89         quick_sort( lt, md-1, data );
90         #pragma omp task
91         quick_sort( md+1, rt, data );
92     }
```

→ Significantly extends Visual Studio debugger capabilities. First OpenMP 3.0 debugger I know of.

ID	State	Type	Team	Parent	# Spawned	Thread	Location
4	Suspended	Implicit, tied	2	0	2	4736	-
9	Suspended	Implicit, tied	1	4	1	4736	C:\Users\ct747764\Documents\demo\qsort_with_tasks\qsc
10	Suspended	Implicit, tied	1	4	0	4360	C:\Users\ct747764\Documents\demo\qsort_with_tasks\qsc
12	Running	Explicit, tied	1	9	0	4736	C:\Users\ct747764\Documents\demo\qsort_with_tasks\qsc
13	Created	Explicit, tied	1	11	0	1844...	C:\Users\ct747764\Documents\demo\qsort_with_tasks\qsc
15	Created	Explicit, tied	1	14	0	1844...	C:\Users\ct747764\Documents\demo\qsort_with_tasks\qsc
19	Running	Explicit, tied	1	17	0	4360	C:\Users\ct747764\Documents\demo\qsort_with_tasks\qsc

Task Spawn Tree

```
Task 4, Suspended, Implicit, tied, Thread 4736, -
├── Task 9, Suspended, Implicit, tied, Thread 4736, C:\Users\ct747764\Documents\demo\qsort_with_tasks\qsort.c:150
│   └── Task 12, Running, Explicit, tied, Thread 4736, C:\Users\ct747764\Documents\demo\qsort_with_tasks\qsort.c:90
│       └── Task 10, Suspended, Implicit, tied, Thread 4360, C:\Users\ct747764\Documents\demo\qsort_with_tasks\qsort.c:150
```

27

Enter for

Computing and
Communication

Motivation

Profiling

Par. Profiling

Debugging

Summary

Agenda

- Motivation and Methodology
- Profiling Tools
- Parallel Profiling Tools
- Debugging and Correctness Tools
- Summary

28

enter for

Computing and

Communication

Motivation

Profiling

Par. Profiling

Debugging

Summary

Summary

- Performance Analysis and Tools require still a lot of Background Knowledge as well as Effort in use
 - But: If you have to deal with parallelization you better forget about printf-debugging right away!
- When HPC on x86 went big with Opteron + Linux, there were hardly any tools for Shared-Memory parallelization
 - Luckily this has changed significantly, with more to come in the future (i.e. Parallel Studio for Linux, Visual Studio 2010, ...),
 - But hardware is still evolving at a fast pace: GPGPUs ...
- HPC can also be read as High Productivity Computing
 - Software should be considered an important ingredient for HPC
 - Are you willing to pay for (good) software tools?!?

29

Center for

Computing and

Communication

Motivation

Profiling

Par. Profiling

Debugging

Summary

The End

Thank you for
your attention!

30