HPC Tools Portfolio:

Tuning and Shared-Memory Parallelization on Windows

Christian Terboven

terboven@rz.rwth-aachen.de

Center for Computing and Communication

RWTH Aachen University

2nd Windows-HPC User Group Meeting
March 31st, TU Dresden, Germany

Agenda

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

Motivation for this Tool Inventory

- Microsoft is pushing Windows HPC Server
  - Opportunity: Integration
    - Operating System + Development Environment + Compiler + Debugger (+ Tools) are all from Microsoft
  - Risk: Windows-HPC is the New Kid on the Block
    - Microsoft might not focus support on legacy (HPC) code
- Interest in Windows-HPC as development platform is high
  - Example from last week's PPCES at Aachen: About 40 percent of the course participants indicated interest in trying the lab exercises on our Windows-HPC platform
    - Switchers? People just interested in *the other side*? People new to HPC / parallel programming? Is the IDE approach appealing?
- What has Windows-HPC to offer *today*?

Methodology: Tuning and Parallelization

- Process of (incremental) parallelization with OpenMP:
  (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

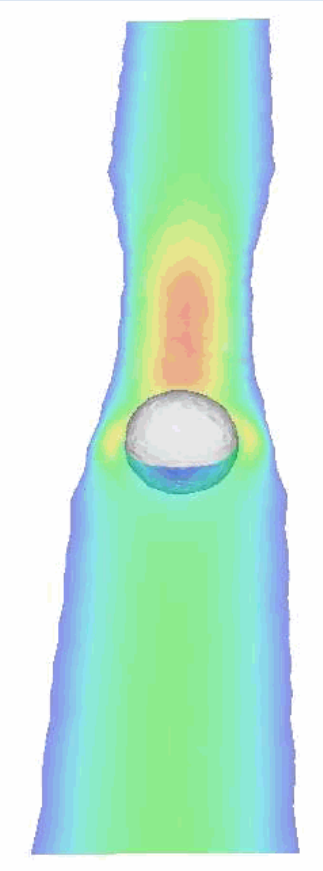
Application used in this Case Study: DROPS

- 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 D_2O (fluid/fluid)



Agenda

- Methodology and Motivation
- Profiling Tools
- Parallelization Tools
- Debugging and Correctness Tools
- 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 three tools:
  - Visual Studio 2008 Profiler
  - Intel Parallel Studio: Amplifier
  - Intel VTune

Visual Studio 2008 Profiler (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 *,u	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>		
std::_Tree<class std::_Tmap_traits<unsigned __int64,class DROPS::FaceCL *,struct std::less<		
DROPS::ParAccurPCG<class DROPS::CompositeMatrixBaseCL<class DROPS::SparseMatBaseCL<		
DROPS::SetupSystem1_P2<class DROPS::ZeroFlowCL>(class DROPS::MultiGridCL const &,class		
DROPS::SetupSystem2_P2P1X<class DROPS::ZeroFlowCL>(class DROPS::MultiGridCL const &,cl		

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

Visual Studio 2008 Profiler (2/2)

- 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 of tuning and / or parallelization decisions.

Missing:
- Simple identification of Hotspots
- Time spent per code line

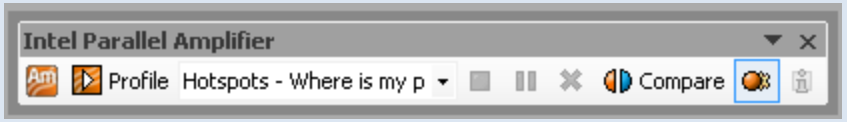
Current View: Call Tree

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
DROPS::LinThetaScheme2PhaseCL<class DROPS::InstatNavierStokes2PhaseP2P1CL<class DROPS::Zerof	847	0
[2 rows folded ending in "DROPS::LinThetaScheme2PhaseCL<class DROPS::InstatNavierStokes2PhaseP2P	856	0
DROPS::LinThetaScheme2PhaseCL<class DROPS::InstatNavierStokes2PhaseP2P1CL<class DROPS::Zerof	856	0
[2 rows folded ending in "DROPS::LinThetaScheme2PhaseCL<class DROPS::InstatNavierStokes2PhaseP2P	20.430	0
DROPS::InstatStokes2PhaseP2P1CL<class DROPS::ZeroFlowCL>::SetupSystem1(class DROPS::MatDescB	1.295	0
DROPS::AdaptFixedPtDefectCorrCL<class DROPS::InstatNavierStokes2PhaseP2P1CL<class DROPS::Zerc	18.818	0

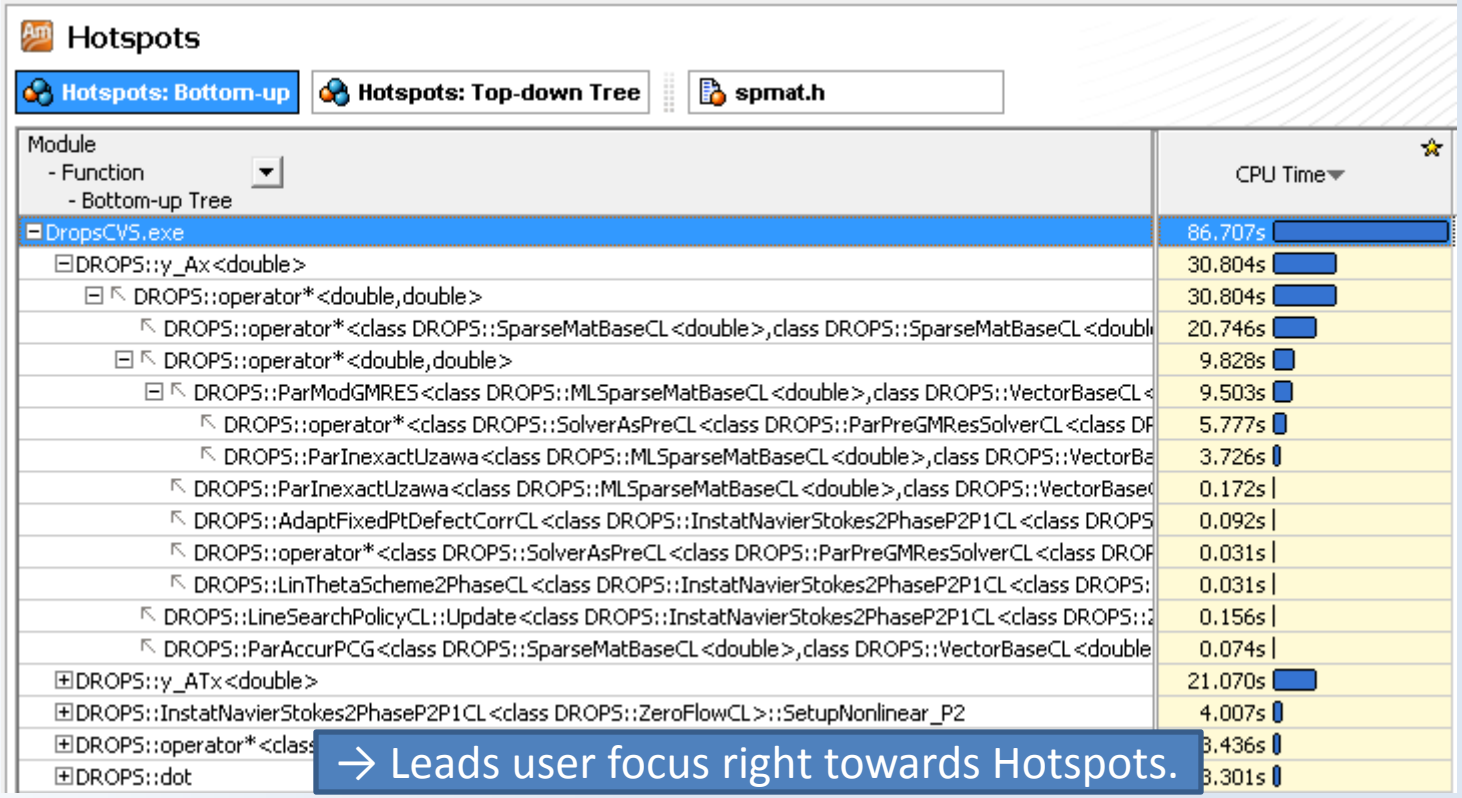
Intel Parallel Studio: Amplifier (1/2)

- Execution time: 87.6 seconds over 77.5 seconds wo/ tool
- Available from within VS:



- Hotspot-based display of analysis result:

→ Easily accessible.



→ Leads user focus right towards Hotspots.

Intel Parallel Studio: Amplifier (2/2)

- Examination of analysis result down to source line level:

```
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 suprise users very often, thus this view should be examined before the first efforts towards parallel-ization are carried out.

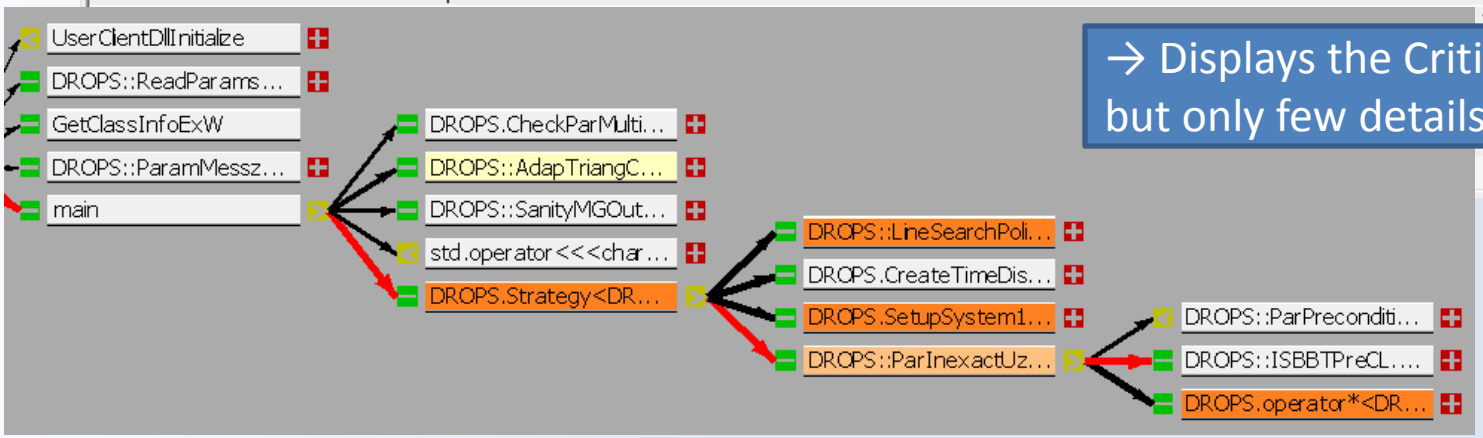
Missing:
- Hardware Counter measurements
- Better representation of call graph

Intel VTune (1/1)

- Execution time:
  - Sampling: Failed for this application on our systems
  - Call Graph Profiling: 77.6 Seconds over 77.5 seconds wo/ tool

- 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	987	22.242	1
MatDescBaseCL<DROPS::MLSparseMatBas...	DROPS::MatDescBaseCL<DROPS::M...	10	4	20	4



→ Displays the Critical Path, but only few details.

Agenda

- Methodology and Motivation
- Profiling Tools
- Parallelization Tools
- Debugging and Correctness Tools
- 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 four tools:
  - Visual Studio 2008 Profiler
  - Intel Parallel Studio: Inspector
  - Intel Thread Profiler
  - Intel VTune

Test: Visual Studio 2008 Profiler (1/1)

- No explicit support for threads or OpenMP:

Functions Causing Most Work

Name	Samples	%
[kernel32.dll]	17.367	100,00
[ntdll.dll]	17.3	
_vcomp::fork_helper	17.3	
_vcomp::ParallelRegion::HandlerThreadFunc(void *,unsigned long)	17.3	
mainomp1	17.2	

Missing:
- Support for threading

→ Just calls to OpenMP runtime functions are presented.

- Comparison of two profiles (serial vs. parallel) is possible:

Comparison Report

Comparison Files
Baseline File: smxv-benchmark090329(1).vsp
Comparison File: smxv-benchmark090329(2).vsp

Comparison Options
Table: Function
Column: Inclusive Samples
Threshold: 1
Apply

Comparison complete.

Comparison Column	Delta	Baseline Value	Comparison Value
drv_init	7	33	40
y_Ax_serial	7	15	22
_vcomp_barrier	1		
y_Ax_omp	-1		
[kernel32.dll]	-2		
[ntdll.dll]	-2		

→ One can derive the improvements, but that is about it.

Intel Parallel Studio: Amplifier (1/1)

- Simpe thread utilization presentation per function:

Concurrency

Concurrency: Bottom-up | Concurrency: Top-down Tree

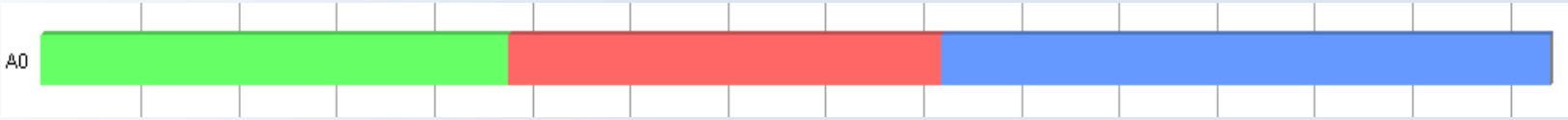
Function - Bottom-up Tree	Module	CPU Time by Utilization (Poor / Ok / Ideal / Over)
r_rtime	smxv-benchmark.exe	0.125s
y_Ax_omp	smxv-benchmark.exe	0.337s
main ← _tmainCRTStartup ← BaseThreadInitThunk ← RtlUserThreac	smxv-benchmark.exe	
mainomp1	smxv-benchmark.exe	
vcomp::fork_helper ← _vcomp::ParallelRegion::HandlerThreadFu	smxv-benchmark.exe	
main ← _tmainCRTStartup ← BaseThreadInitThunk ← RtlUserThr	smxv-benchmark.exe	
y_Ax_serial	smxv-benchmark.exe	0.077s
drv_init	smxv-benchmark.exe	0.051s
main	smxv-benchmark.exe	0.031s
drv_initomp1	smxv-benchmark.exe	1.425s
_CxxSetUnhandledExceptionFilter	smxv-benchmark.exe	0.000s

→ Only very basic information. More details with respect to load balancing (for example) are very much wanted.

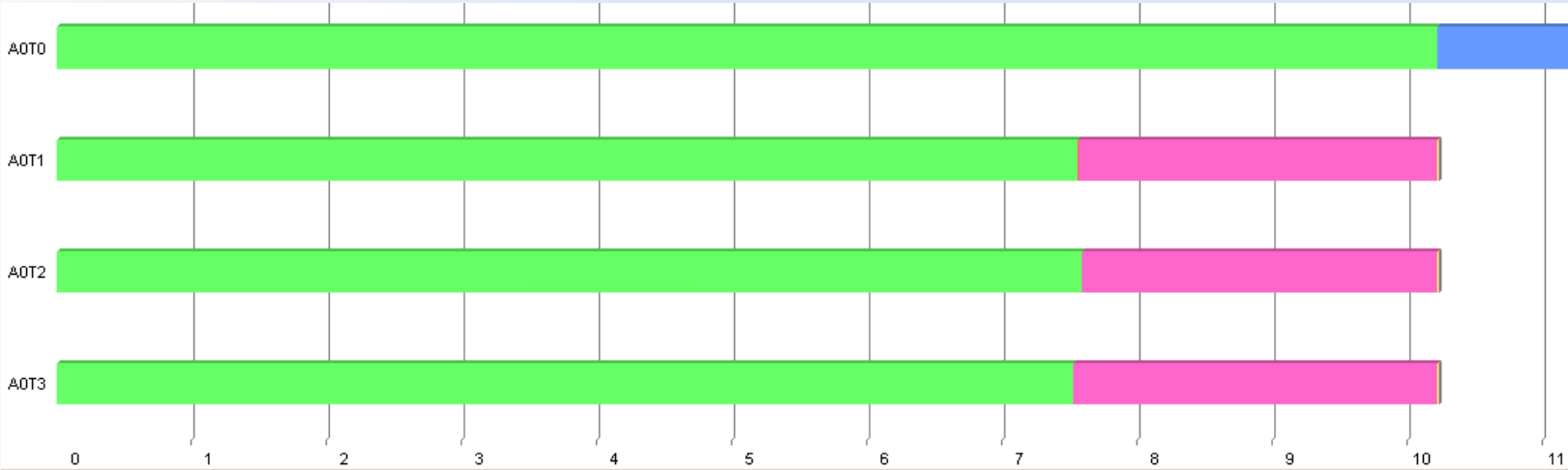
Missing:
- Expert information
- More fine-grained details

Intel Thread Profiler (1/1)

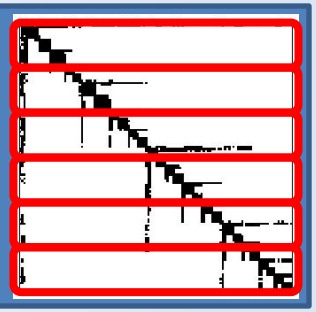
- Load balance statistics - whole program:



- and per parallel region:



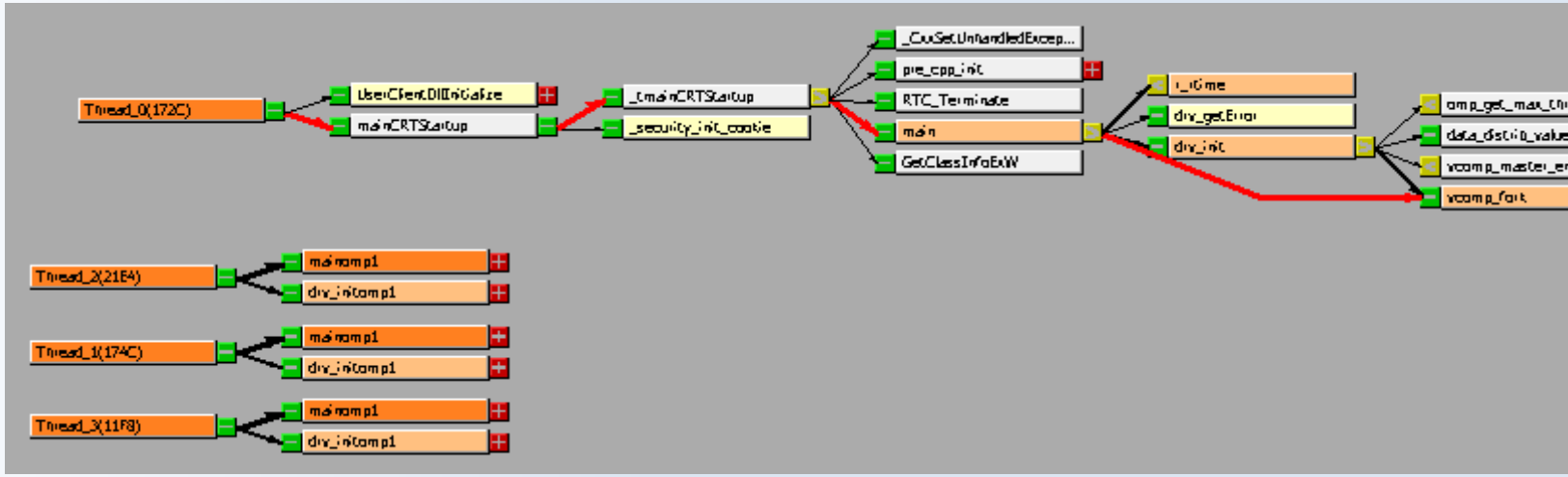
→ Allows for mapping of dataset load characteristics to execution profile:



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

Intel Thread VTune (1/1)

- Call graph profile for multiple threads:



→ In many cases of only little use.

Agenda

- Methodology and Motivation
- Profiling Tools
- Parallelization Tools
- Debugging and Correctness Tools
- 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 eleminate any kind of bug in a parallel program

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

Visual Studio 2008 Debugger (1/1)

- Visual Studio 2008 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:

Threads

	ID	Name	Location	Priority	Suspend
➡	7324	jacobiomp1	jacobiomp1	Normal	0
⏸	5424	jacobiomp1	jacobiomp1	Normal	0
⏸	5740	jacobiomp1	jacobiomp1	Normal	0

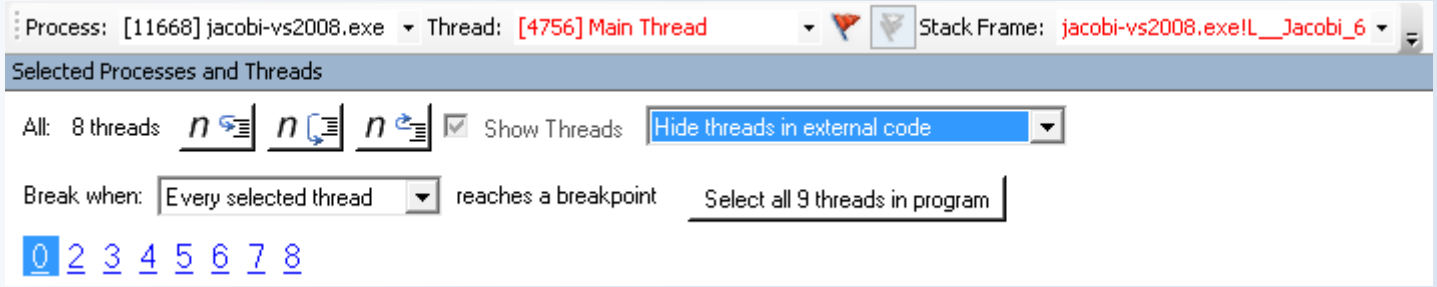
Autos | Locals | Processes | Threads | Modules | Watch 1

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

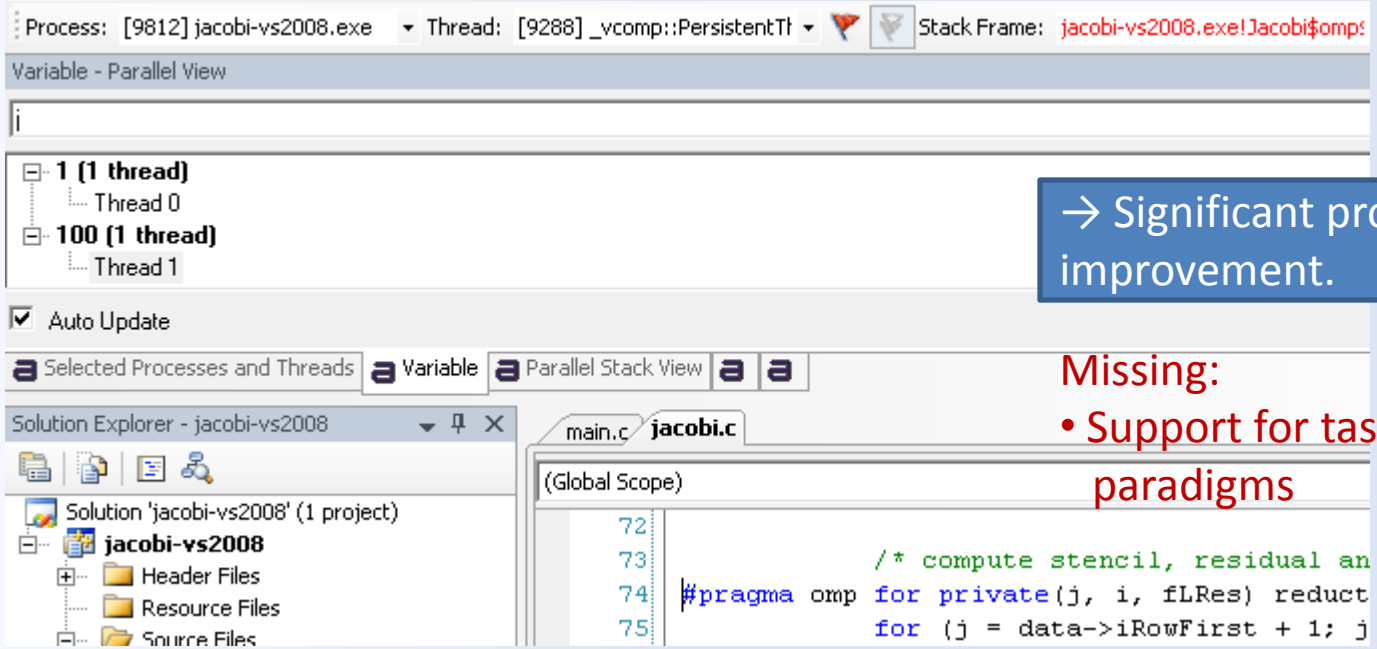
Missing:
- Better control of thread groups
- Laminated view of private variables

Visual Studio 2008 Debugger w/ DDTlite (1/1)

- Individual thread grouping and switching:



- Display of variable values per thread:

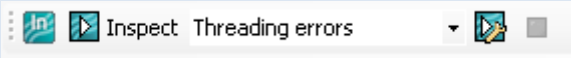


→ Significant productivity improvement.

Missing:
- Support for task-based paradigms

Intel Parallel Studio: Inspector (1/1)

- **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.
  - Non-deterministic occurrence



→ Easily accessible.

- (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 confl memory write at "main.c":196 (flow		
2	6	Write -> Write data-race		Memory write at "jacobi.c":66 confl memory write at "jacobi.c":69 (outp		
2	7	Read -> Write		Memory write at "jacobi.c":66 conflicts with a prior	69	False

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

Intel Parallel Studio: Composer (1/1)

- 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.

Tasks

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\qs
10	Suspended	Implicit, tied	1	4	0	4360	C:\Users\ct747764\Documents\demo\qsort_with_tasks\qs
12	Running	Explicit, tied	1	9	0	4736	C:\Users\ct747764\Documents\demo\qsort_with_tasks\qs
13	Created	Explicit, tied	1	11	0	1844...	C:\Users\ct747764\Documents\demo\qsort_with_tasks\qs
15	Created	Explicit, tied	1	14	0	1844...	C:\Users\ct747764\Documents\demo\qsort_with_tasks\qs
19	Running	Explicit, tied	1	17	0	4360	C:\Users\ct747764\Documents\demo\qsort_with_tasks\qs

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

Agenda

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

Summary (1/2)

Task	Windows		Linux
	Microsoft	3rd Party	3rd Party
Compiler	C/C++	Intel C/C++ & F95	GCC C/C++ & F95 Intel C/C++ & F95 Sun C/C++ & F95
IDE	Visual Studio 2008	Eclipse	Eclipse, Kdevelop, ...
(Parallel) Profiling	Visual Studio 2008	Intel Parallel Studio Intel VTune	gprof Intel VTune Sun Studio
Parallelization		Intel Parallel Studio Intel Threading Tools	Intel Threading Tools, but wo/ GUI Sun Studio
(Parallel) Debugging	Visual Studio 2008	Intel Parallel Studio Allinea DDTlite	TotalView Allinea DDT (support for tasks)

- Black = well-suited tool for task presented, Gray = tool is not well-suited

Summary (2/2)

- Windows offers a very complete development environment
  - Centered around Visual Studio 2008 → **Integrated** Dev. Env.
  - Market place and opportunities for 3d Parties
  - Would look different without Intel's software efforts though
- HPC can also be read as High Productivity Computing
  - IDE with good support for performance analysis and Shared-Memory parallelization is missing on other platforms
  - Software should be considered an important ingredient for HPC
- Software environment on Windows does not come for free
  - Are you willing to pay for (good) software tools?!?
- Microsoft is promising much improved support for multi-core parallelization with Visual Studio 2010 (see CTP or Beta1) ...

The End

Thank you for

your attention!

WinHP3C:

http://www.rz.rwth-aachen.de/winhp3c