

Optimize code for Intel Xeon Phi

Discovering bottlenecks without pain



www.allinea.com





- Introduction
- Allinea MAP and optimizing for Xeon Phi
- Conclusion



Allinea Unified environment

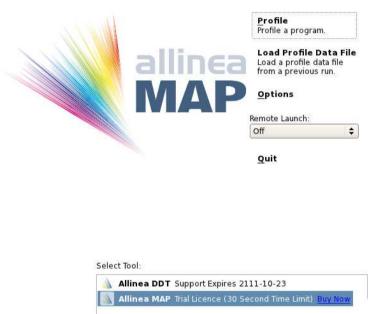
- A modern integrated environment for HPC developers
- Supporting the lifecycle of application development and improvement
 - Allinea DDT : Productively debug code
 - Allinea MAP : Enhance application performance
- Designed for productivity
 - Consistent easy to use tools
 - Enables effective HPC development
- Improve system usage
 - Fewer failed jobs
 - Higher application performance



www.allinea.com

Allinea MAP Increase application performance

- Parallel profiler designed for:
 - C/C++, Fortran
 - Multiprocess code
 - Interdependent or independent processes
 - Multithreaded code
 - Monitor the main threads for each process
 - Accelerated codes
 - GPUs, Intel Xeon Phi
- Improve productivity :
 - Helps you detect performance issues quickly and easily
 - Tells you immediately where your time is spent in your source code
 - Helps you to optimize your application efficiently





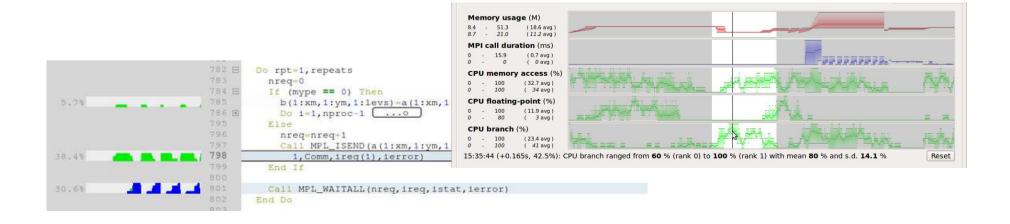
Allinea MAP Find performance issues quickly



- Look at the entire application on real data sets
 - Visualize the entire run at full scale, not just reduced sets
 - Zoom in to explore iterations, functions and loops

• Understand the nature of bottlenecks

- Source code viewer pinpoints bottleneck locations
- CPU, MPI, I/Os and memory metrics identify the cause



Allinea MAP The missing link in HPC tools

• Unique profiling methodology (even on Intel Xeon Phi)

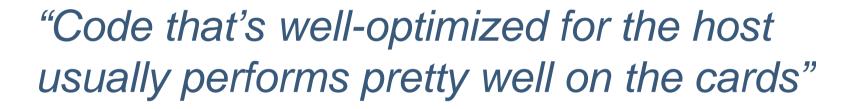
- No need to instrument your code
- Just 5% wall-clock overhead
- Small output files (10-20Mb is typical)

• Benefit:

- 80% of the profiling activity can now be done by end-users
- Helps to focus on the right bit of the application
- Brings more value to the expertise of application support



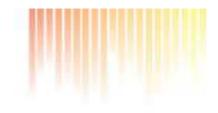
Optimizing for the Xeon Phi Where do you start?

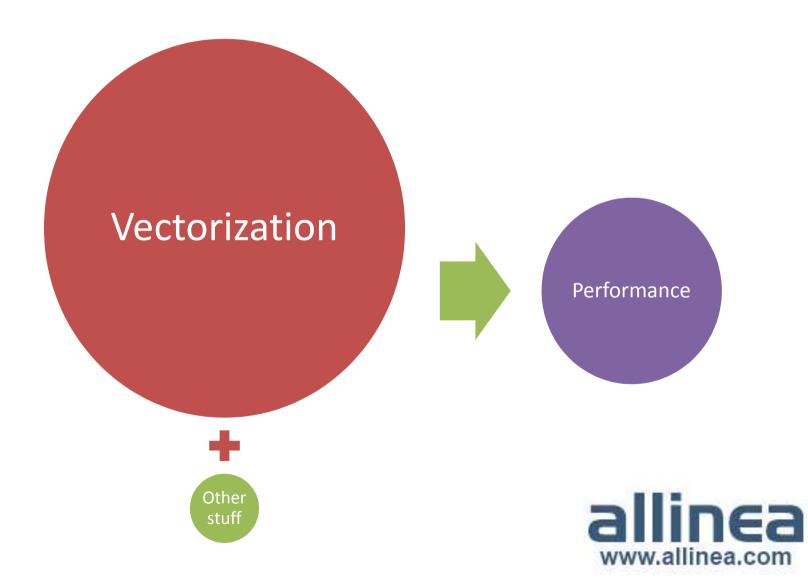


- Pretty much everyone



Optimizing for the Xeon Phi But what matters?





Optimizing for the Xeon Phi Is my code well-vectorized?



mg.f(2432): (col. 10) remark: loop was not vectorized: not inner loop. mq.f(2431): (col. 7) remark: loop was not vectorized: not inner loop. mg.f(993): (col. 13) remark: LOOP WAS VECTORIZED. mq.f(992): (col. 10) remark: loop was not vectorized: not inner loop. mg.f(991): (col. 7) remark: loop was not vectorized: not inner loop. mg.f(243): (col. 7) remark: loop was not vectorized: existence of vector depende nce. mg.f(993): (col. 13) remark: LOOP WAS VECTORIZED. mq.f(992): (col. 10) remark: loop was not vectorized: not inner loop. mg.f(991): (col. 7) remark: loop was not vectorized: not inner loop. mg.f(753): (col. 13) remark: loop was not vectorized: vectorization possible but seems inefficient. mg.f(762): (col. 13) remark: loop was not vectorized: vectorization possible but seems inefficient. mg.f(749): (col. 10) remark: loop was not vectorized: not inner loop. mq.f(746): (col. 7) remark: loop was not vectorized: not inner loop. mg.f(993): (col. 13) remark: LOOP WAS VECTORIZED. mq.f(992): (col. 10) remark: loop was not vectorized: not inner loop. mq.f(991): (col. 7) remark: loop was not vectorized: not inner loop. mq.f(2255): (col. 16) remark: loop was not vectorized: existence of vector depen dence. mq.f(2254): (col. 13) remark: loop was not vectorized: not inner loop. mq.f(2251): (col. 7) remark: loop was not vectorized: not inner loop. mg.f(2433): (col. 13) remark: LOOP WAS VECTORIZED. mq.f(2433): (col. 13) remark: loop was not vectorized: not inner loop. mq.f(2432): (col. 10) remark: loop was not vectorized: not inner loop. mq.f(2431): (col. 7) remark: loop was not vectorized: not inner loop. mg.f(2433): (col. 13) remark: LOOP WAS VECTORIZED. mg.f(2433): (col. 13) remark: loop was not vectorized: not inner loop. mg.f(2432): (col. 10) remark: loop was not vectorized: not inner loop. mg.f(2431): (col. 7) remark: loop was not vectorized: not inner loop. mg.f(527): (col. 7) remark: loop was not vectorized: nonstan<u>dard loop is not a v</u> ectorization candidate. mg.f(552): (col. 7) remark: loop was not vectorized: nonstandard loop is not a v ectorization candidate. mg.f(1150): (col. 7) remark: loop was not vectorized: loop was transformed to me mset or memcpy. mg.f(1150): (col. 7) remark: l<u>oop was not vectorized: loop was transformed to me</u> mset or memcpy. mg.f(1645): (col. 7) remark: loop was not vectorized: loop was transformed to me mset or memcpy. $ma f(1655) \cdot (col)$ 7) remark: loop was not vectorized: loop was transformed to

... maybe?



Optimizing for the Xeon Phi Is my code well-vectorized?



| <u>V</u> iew Se <u>a</u> rch <u>W</u> ind | - Allinea MAP v dow <u>H</u> elp | | | ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ | | |
|---|---|--|---|--|--------------------------------------|----------------------|
| ofiled: slow_f on 8 pro | cesses Started: | Thu Mar 14 14:03:16 201 | 13 Runtime: 30s Time in MPI: 37 % | | | Hide Metrics |
| Memory usage (M) | | | | | | |
| | (17.9 avg) (22.2 avg) — | | | | | |
| MPI call duration (| | | | | | |
| 0 - 5,567.9 (| 642.7 avg) | | | | | |
| | 0 avg) | | | | | |
| | SS (%) 2.2 avg) 10 avg) | | | Longitude West | | |
| CPU floating-point | | | | an an an an air an | | |
| 0 - 100 (| 35.6 avg) 44 avg) | ment and | my many in | | | |
| CPU floating point | 27.7 avg) | | - Martin | | | |
| 0 - 10 (14:03:31-14:03:36 (ra | 0 avg) | 5% of total): Mean Memory | y usage 22.2 M; Mean MPI call duration 0.0 |) ms; Mean CPU memory - | access 10.4 %; Mean CPI | U floating-poi Reset |
| 0 - 10 (.4:03:31-14:03:36 (ra | ange 4.928s, 16. 100 101 | ! inefficient | y usage 22.2 M; Mean MPI call duration 0.0 |) ms; Mean CPU memory - | access 10.4 %; Mean CPI | U floating-poi Reset |
| 0 - 10 (14:03:31-14:03:36 (ra slow.f90 🗶 | ange 4.928s, 16. 100 | ! inefficient do l=1,500 | y usage 22.2 M; Mean MPI call duration 0.0 |) ms; Mean CPU memory - | access 10.4 %; Mean CPI | U floating-poi Reset |
| 0 - 10 (| 100 101 102 103 104 | ! inefficient do 1=1,500 do i=1,2000 do j=1,2000 | y usage 22.2 M; Mean MPI call duration 0.0 |) ms; Mean CPU memory - | access 10.4 %; Mean CPI | U floating-poi Reset |
| 0 - 10 (L4:03:31-14:03:36 (ra slow.f90 X | 100 101 102 103 104 105 | ! inefficient do l=1,500 do i=1,2000 do j=1,2000 x=1 | y usage 22.2 M; Mean MPI call duration 0.0 |) ms; Mean CPU memory - | access 10.4 %; Mean CPI | U floating-poi Reset |
| 0 - 10 (L4:03:31-14:03:36 (ra slow.f90 X | 100 101 102 103 104 105 106 | <pre>! inefficient do l=1,500 do i=1,2000 do j=1,2000</pre> | |) ms; Mean CPU memory - | access 10.4 %; Mean CPI | U floating-poi Reset |
| 0 - 10 (L4:03:31-14:03:36 (ra slow.f90 ★ | 100 101 102 103 104 105 106 106 107 108 | <pre>! inefficient do l=1,500 do i=1,2000 a do j=1,2000 x=i y=j a(i,j)=x* end do</pre> | |) ms; Mean CPU memory - | access 10.4 %; Mean CPI | U floating-poi Reset |
| 0 - 10 (14:03:31-14:03:36 (ra slow.f90 ★ | 100 101 102 103 104 105 106 105 106 | <pre>! inefficient do l=1,500 do i=1,2000 x=i y=j a(i,j)=x** end do end do</pre> | |) ms; Mean CPU memory - | access 10.4 %; Mean CPI | U floating-poi Reset |
| 0 - 10 (L4:03:31-14:03:36 (ra slow.f90 ★ .18 .98 | 100 101 102 103 104 105 106 107 108 109 110 | <pre>! inefficient do l=1,500 do i=1,2000 x=i y=j a(i,j)=x* end do end do end do</pre> | |) ms; Mean CPU memory - | access 10.4 %; Mean CPI | U floating-poi Reset |
| 0 - 10 (L4:03:31-14:03:36 (ra slow.f90 ★ -18 -98 | 100 101 102 103 104 105 106 107 108 109 110 | <pre>! inefficient do l=1,500 do i=1,2000 x=i y=j a(i,j)=x** end do end do</pre> | j |) ms; Mean CPU memory - | access 10.4 %; Mean CPI | |
| 0 - 10 (L4:03:31-14:03:36 (ra slow.f90 ★ .18 .98 | 100 101 102 103 104 105 106 107 108 109 110 | <pre>! inefficient do l=1,500 do i=1,2000 x=i y=j a(i,j)=x* end do end do end do</pre> | j |) ms; Mean CPU memory - | access 10.4 %; Mean CPI | |
| 0 - 10 (L4:03:31-14:03:36 (ra slow.f90 X .18 .08 | ange 4.928s, 16. 100 101 102 103 105 106 107 108 109 110 Files Parallel | <pre>! inefficient do 1=1,500 do i=1,2000 x=i y=j a (i, j) = x** end do end do end do Stack View</pre> | j |) ms; Mean CPU memory - | | |
| 0 - 10 (L4:03:31-14:03:36 (ra slow.f90 ★ .18 .98 | ange 4.928s, 16. 100 101 102 103 104 105 106 107 108 109 110 Files Parallel S MPI | <pre>! inefficient do l=1,500 do i=1,2000 x=i y=j a(i,j)=x** end do end do end do Stack View Function(s) on line</pre> | 1 Source program slow call stride |) ms; Mean CPU memory - | Positi slow.f slow.f | on 90:1 90:11 |
| 0 - 10 (L4:03:31-14:03:36 (ra slow.f90 X .18 .08 | ange 4.928s, 16. 100 101 102 103 104 105 106 107 108 109 110 Efiles Parallel S MPI | <pre>! inefficient do 1=1,500 do i=1,2000 x=i y=j a (i, j) = x** end do end do end do Stack View</pre> | j Source program slow |) ms; Mean CPU memory - | Positi slow.f slow.f slow.f | 0n 90:1 |

Optimizing for the Xeon Phi Is my code well-vectorized?

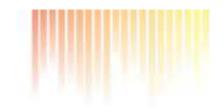


| 0 - | 50 | (10 avg) | |
|-----------|------------|------------------------|--|
| CPU flo | ating-po | int (%) | And the state of the second state of the secon |
| | 100 100 | (35.5 avg) (44 avg) | |
| CPU flo | ating po | int vector (%) | |
| 0 - | 100 10 | (27.7 avg) (0 avg) | A many and the second s |
| 14:03:31- | | | 16.5% of total): Mean Memory usage 22.2 M; Mean MPI call duration 0.0 ms; Mean CPU memory acce |

Not in this loop (16.5% of total time)

| 1 | 02 | do 1=1,500 |
|-------|-----|-------------|
| <0.1% | 03 | do i=1,2000 |
| 10.9% | 04 | do j=1,2000 |
| 1 | .05 | x=i |
| 1 | 06 | y=j |
| 89.08 | .07 | a(i,j)=x*j |
| 1 | 80. | end do |
| 1 | 09 | end do |
| 1 | 10 | end do |
| | 4.4 | |

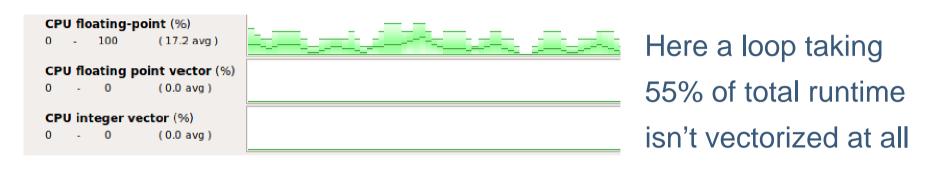
Optimizing for the Xeon Phi Non-obvious tradeoffs

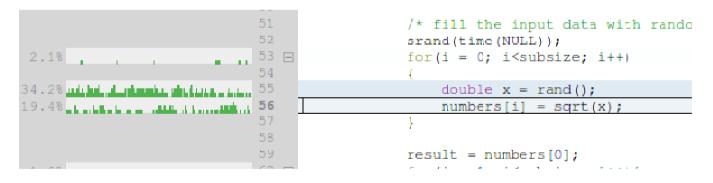


| 😣 🗆 🗊 sqrtmax-fusion.map - Alline | a MAP v4-1-BRANCH | | |
|--|---|--|---|
| <u>File View Search W</u> indow <u>H</u> elp | | | |
| Profiled: sqrtmax-fusion on 4 processes S | itarted: Mon Mar 11 13:20: | 53 2013 Runtime: 2s Time in MPI: 40 % | Hide Metrics |
| Memory usage (M) | | | |
| MPI call duration (ms) 0 - 2.3 (0.4 avg) | | | |
| CPU floating-point (%) 0 - 100 (17.2 avg) | الأليني | محمقهم بالمأصلان بالمباد بالمقا | |
| CPU floating point vector (%) 0 - 0 (0.0 avg) | | | |
| CPU integer vector (%) 0 - 0 (0.0 avg) | | | |
| 13:20:53-13:20:54 (range 1.671s): Mea | n Memory usage 7.2 M; Me | an MPI call duration 0.4 ms; Mean CPU floating-point 17.2 %; Mean CPU floating point vector | 0.0 %; Mean CPU inte Reset |
| 🛛 sqrtmax-fusion.c 🗶 | | | |
| 50 51 52 2.1% | srand(ti | the input data with random numbers ourselves */ me(NULL)); 0; i <subsize; i++)<="" th=""><th></th></subsize;> | |
| 34.2% | | <pre>le x = rand();</pre> | |
| 19.4% 56 | numb | ers[i] = sqrt(x); | |
| 58 59 1.0% | for(i = | numbers[0]; 1; i <subsize; i++){<="" th=""><th></th></subsize;> | |
| Input/Output Project Files Parallel St | tack View | | |
| Parallel Stack View | | | ē X |
| Total Time 🔺 MPI | Function(s) on line | Source | Position |
| 34.2% odd. bill of a standak of a farmer in a standak of a standard stand | main frand, rand@plt> mPl_Recv MPl_Recv | <pre>int main(int argc, char *argv[]) { double x = rand(); MPI_Recv(&result, 1, MPI_DOUBLE, i, 0, MPI_COMM_WORLD, &status); numbers[i] = sqrt(x); </pre> | sqrtmax-fusion.c:9 sqrtmax-fusion.c:55 sqrtmax-fusion.c:36 sqrtmax-fusion.c:56 |
| 8.8% | MPI_Recv MPI_Recv MPI_Recv MPI_Send | <pre>MPI_Recv(&subsize, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status); MPI_Recv(numbers, subsize, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status); MPI_Send((numbers+sIndex+(subsize*(i-1))), subsize, MPI_DOUBLE, i, 0, MPI_COMM. if(result < numbers[i])</pre> | sqrtmax-fusion.c:48 sqrtmax-fusion.c:49 |
| / X %A | > | | H 34a9cd1d0317 Jun 10 2013 |

Optimizing for the Xeon Phi Non-obvious tradeoffs



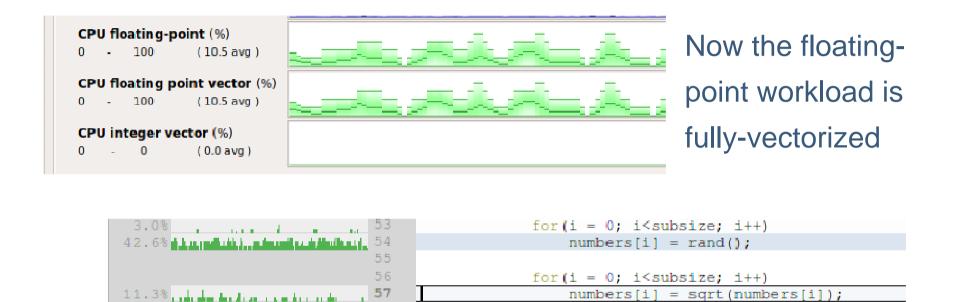




Taking the unvectorizable rand() out of the loop allows the sqrt workload to be fully-vectorized – reverse loop fusion!

Optimizing for the Xeon Phi Non-obvious tradeoffs





But all the time is being spent in the random number generation, so that's what really needs to be optimized

L 59 E

1.7%

result = numbers[0];

for (i = 1; i<subsize; i++) {

Optimizing for the Xeon Phi Know your tools



Random Number Function Vectorization

Submitted by Ronald W Green ... on Fri, 09/07/2012 - 16:31

Categories: Intel® Many Integrated Core Architecture , Vectorization , Intel® C++ Compiler , Intel® Fortran Compiler , C/C++ , Fortran , Developers , Linux* , Advanced

Tags: Random Number Function Vectorization

Drand48 Vectorization in C/C++ Goodman, Steve9700.000000000 Compiler Methodology for Intel® MIC Architecture

Vectorization Essentials, Random Number Function Vectorization

The Intel 13.0 Product Compiler now supports random number auto- vectorization of the drand48 family of random number functions in C/C++ and RANF and Random_Number functions in Fortran. Vectorization is supported through the Intel Short Vector Math Library (SVML).

Supported C/C++ Functions:

double drand48(void); double erand48(unsigned short xsubi[3]); long int lrand48(void); long int prand48(unsigned short xsubi[3]);

Replace rand() with Intel's vectorized version and re-fuse the loop to retain temporal cache locality benefits

Optimizing for the Xeon Phi The full picture



| •••• | oc/training/map-introdu | uction/2-cpu-optimization/solution/sqrtmax-ffastmath.map - Allinea MAP v4-i-BR/ | ANCH |
|---|----------------------------------|--|--|
| <u>File View Search Window H</u> elp | | | |
| Profiled: sqrtmax-vector on 4 processes | Started: Mon Mar 11 12:17 | ':19 2013 Runtime: 2s Time in MPI: 38% | Hide Metrics |
| Memory usage (M) 5.4 - 7.7 (7.2 avg) | | | |
| MPI call duration (ms) 0 - 3.1 (0.5 avg) | | | |
| CPU floating-point (%) 0 - 100 (10.5 nvg) | فجريجو | kala laka lak | |
| CPU floating point vector (%) 0 - 100 (10.5 avg) | | k ala likki li k | |
| CPU integer vector (%) 0 - 0 (0.0 avg) | | | |
| 12:17:19-12:17:20 (range 1.987s): Nea | an Memory usage 7.2 M; Me | ean MPI call duration 0.5 ms; Mean CPU floating-point 10.5 %; Mean CPU floating point vector 10 | .5 %; Mean CPU int Reset |
| © sqrtmax-workergen.c 🗙 | | | |
| 52 53.0% 12.6% | for(i - numb | <pre>ime(NULL);; 0; i<subsize; i++)<br="">sers[i] = rand();</subsize;></pre> | |
| 56 11.3% | | 0; l≺subsize; i++) pers[i] - sgrt(numbers[i]); | |
| 58 | result = | = numbers[0]; | |
| 1.7% 59 E | for (1 = | 1; INSUBSIZE; Itt)(result < numbers[i]) | |
| | tack View | | |
| Input/Output Project Files Parallel S arallel Stack View | | | |
| italier stack view | Function(s) on line | Course | Position |
| star nine · Min | i main | Source int main(int args, char *argw[]); | sqrtmax-workergen.c:9 |
| 42.6%)) | ∉ rand, rand@plt> | <pre>numbers[1] = rand(); MP1_Recv(&result, 1, MP1_DJUBLE, 1, 0, MP1_COMM_WORLD, &status);</pre> | sqrtmax-workergen.c:54 sqrtmax-workergen.c:36 |
| 11.3%, | E MPI_Recv F MPI_Recv | <pre>numbers'ill = sqtf(numbers(il): MPI_Recv(subsize, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status); MPI_Recv(numbers, subsize, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status); for(i = 0, i<cubsize, iii)<="" pre=""></cubsize,></pre> | sqrtmax-workergen.c:57 sqrtmax-workergen.c:48 sqrtmax-workergen.c:49 sqrtmax workergen.c:53 |
| 2.6% | MPI_Send 2 others | <pre>if(result < numbers[1]) Pl_gend(numberstallndex+(subsize*(i-1))), subsize, MPI_DOUBLE, i, 0, MPI_COMM_W_ for(i = 1; i<subsize; i++){<="" pre=""></subsize;></pre> | sqrtmax-workergen.c:60 sqrtmax-workergen.c:33 sqrtmax-workergen.c:59 |
| | | Allinea MAP y4-1-DRANCH 3 | 4a9cd1d0317 Jun 10 201 |

You need to see the full picture to spot these tradeoffs – Allinea MAP shows you the way

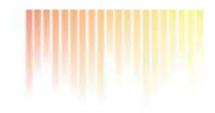
Optimizing for the Xeon Phi Running on the card



| Profiled: wave-c on 4 p | processes Started: F | Fri Apr 5 17:34:54 2013 | untime: 35s Time in MPI: 4% | Hide Metrics |
|--------------------------------------|---|-------------------------|--|---|
| Memory usage (N | | | | |
| 10.6 - 23.3 | (14.7 avg) | | | |
| MPI call duration | (ms) (0.3 avg) | | | Wileddille i Hiterdi Mantatara a consta Istoratakik |
| | (0.5 0.9) | | | |
| CPU floating-poir 0 - 50 | nt (%) (10.2 avg) | | عبرو بالمنافع بالمنافع بالمنافع بالمنافع بالمنافع المنافع المنافع المنافع والمنافع المنافع الم | المالي المسلم الم المالي المسلم الم الم |
| 17:34:54-17:35:29 (| range 34.641s): Mea | an Memory usage 14.7 M; | ean MPI call duration 0.3 ms; Mean CPU floating-point 10.2 %; | Reset |
| 🕫 wave.c 🕱 | | | | |
| | 195 | | | |
| | | | e points along line */ | |
| 3.53 | | for (j | 1; j <= npoints; j++) | |
| | 199 | 1 /* | lobal endpoints */ | |
| 4.83 | 200 | if | (first + j - 1 == 1) (first + j - 1 == tpoints)) | |
| 1.91 | 201 | . 1 | newval[j] = 0.0; | |
| 66.81 010-000 000000000 | 202 000-000-000 203 | els | io math(j); | |
| NEXTRACTOR . | 204 | 1 | | |
| 3.98 | 205 日 | for (j | 1; j <= npoints; j++) | |
| 11.45 | 206 | | al[j] = values[j]; | |
| 3,11 | | | s[j] = newval[j]; | |
| | | 1 | | - |
| | | | | |
| | 212 | allt = (end.tv | ec - start.tv sec) * 1000000 + (end.tv nsec - start.tv nsec) / 1000; | |
| | 213 | double calculat | on_rate = ((double)tpoints / (double)allt) * iterations; // in million points per second | |
| | 214 | | rintf("points / second: %.1fM (%.1fM per process)\n", calculation_rate, calculation_rate / ntask) | F |
| | | | <pre>y = (double)(allt - communication usec) / (double)allt; mpute / communicate efficiency: %d%% %d%% %d%% n", (int)(100 * efficiency + 0.5));</pre> | |
| | 217 | } | space / communicate efficiency, succes / success, fine, fine, first efficiency + 0.577, | |
| | 218 | 2. | | |
| | 219 🖽 | /* | */_) | |
| Input/Output Project | ct Files Parallel Sta | ack View | 440000000 | C |
| Parallel Stack View | CU Files Farallel Sta | | | 8 |
| Total Time | A MPI | Function(s) on line | Source | Position |
| | | 🖻 main | t. | wave.c:282 |
| C.C. Operative militariant at annual | and the second second second | 🗏 update | update(left, right); | wave.c:308 |
| 66.8% http://doi.org/10.000 | Sector Reading | | <pre>do_math(j); oldval(j) = values(j);</pre> | wave.c:203 wave.c:207 |
| 4.8% | | 1 - 1 | if ((first + j - 1 == 1) (first + j - 1 == tpoints)) | wave.c:200 |
| 3.9% | | | <pre>for (j = 1; j <= npoints; j++)</pre> | wave.c:205 |
| 3.5% | and the second state and the | | <pre>for (j = 1; j <= npoints; j++)</pre> | wave.c:197 |
| 3.1% | | E MOL DA | values[j] = newval[j]; | wave.c:208 |
| 2.7% | **** databased 2.7% | MPI_Recv | <pre>MPI_Recv(&values[0], 1, MPI_DOUBLE, left, E_LtoR, MPI_COMM_WORLD, newval[j] = 0.0;</pre> | wave.c:175 wave.c:201 |
| 1.9% | 1.4% | H MPI Recv | MPI_Recv(&values[npoints+1], 1, MPI_DOUBLE, right, E_RtoL, | wave.c:201 wave.c:181 |
| | the second se | | 1999년 7월 20일 - 1999년 1999년 1999년 1999년 1991년 1 | |

Allinea MAP runs with full metrics on Xeon Phi cards!

Optimizing for the Xeon Phi Running on the card



| Profiled: wave-c on 4 proc | cesses Started: | Fri Apr 5 17:34:54 2013 | untime: 35s Time in MPI: 4% | Hide Metrics |
|-------------------------------------|-----------------------------------|-------------------------|--|---|
| | - | | | |
| Memory usage (M) 10.6 - 23.3 (1 | 14.7 avg) | | | |
| | | | | entered in terms of first sector of the sector of the |
| MPI call duration (m 0 - 12.8 (0 | 15) 0.3 avg) | | | |
| CPU floating-point (| (96) | - | | |
| |).2 avg) | unden auf | التلق بقر ومقدمينين والجريفية والترو ويتفر والتكفيك والتقار الشاري ويتبال | When a she is a she was to |
| 17:34:54-17:35:29 (ran | nge 34.641s): Me | an Memory usage 14.7 M; | lean MPI call duration 0.3 ms; Mean CPU floating-point 10.2 %; | Reset |
| | | | | |
| © wave.c 🗶 | 1.95 | | | F |
| | 195 | /* upda | e points along line */ | |
| 3.5% | | for (j | 1; j <= npoints; j++) | |
| | 198 199 | 1 /* | lobal endpoints */ | |
| 4.83 | 200 | | (first + j - 1 == 1) (first + j - 1 == tpoints)) | |
| 1.91 | 201 | | newval[j] = 0.0; | |
| 66.81 0-0 | | el: | do math(j); | |
| | 204 | 1 | | |
| 3.91 | | tor (j | 1; j <= npoints; j++) | |
| 11.45 | 207 | olo | al[j] = values[j]; | |
| 3,11 | 208 | val | es[j] = newval[j]; | |
| | 209 210 | 1 | | |
| | 211 | 1 | | |
| | | allt = (end.tv | ec - start.tv_sec) * 1000000 + (end.tv_nsec - start.tv_nsec) / 1000; | |
| | 213 214 | | on rate = ((double)tpoints / (double)allt) * iterations; // in million points per second rintf("points / second: %.1fM (%.1fM per process)\n", calculation rate, calculation rate / | ntask): |
| | | double efficier | y = (double) (allt - communication usec) / (double) allt; | in case of y |
| | 216 | reduce_print("o | <pre>mpute / communicate efficiency: %d%% %d%% %d%% \n", (int)(100 * efficiency + 0.5));</pre> | |
| | 217 218 | 1 | | |
| | 219 Œ | } /* | ······ | |
| nput/Output Project Fi | iles Parallel St | ack View | E. 201 | t |
| arallel Stack View | nes raraier se | | | 0 |
| otal Time | ▲ MPI | Function(s) on line | Source | Position |
| inne | MIL | E main | Source | wave.c:282 |
| | | Eupdate | update(left, right); | wave.c.202 wave.c:308 |
| 66.8% Linddon of Accord | | | do_math(j); | wave.c:203 |
| 11.4% | ar - 10 halfs - 10 ar an abor - 1 | | oldva[[]] = values[]]; | wave.c:207 |
| 4.8% | | | <pre>if ((first + j - 1 == 1) (first + j - 1 == tpoints)) for (j = 1; j <= npoints; j++)</pre> | wave.c:200 wave.c:205 |
| 3.5% | *** *** | | for (j = 1; j <= npoints; j++) | wave.c:205 wave.c:197 |
| 3.1% | | | values[j] = newval[j]; | wave.c:208 |
| 2.7% | | MPI_Recv | MPI_Recv(&values[0], 1, MPI_DOUBLE, left, E_LtoR, MPI_COMM_WORLD, | wave.c:175 |
| 1.9% | | 1 | newval[j] = 0.0; | wave.c:201 |

This makes it easy to compare and learn versus the host

1.4%

1.4%

MPI Recv

MPI_Recv(&values[npoints+1], 1, MPI_DOUBLE, right, E_RtoL,

wave.c:181





- Allinea tools are the premier Xeon Phi development environment
 - See at a glance which loops to vectorize
 - Full profiling metrics available on the Xeon Phi cards
 - Unified interface with Allinea DDT keeps you productive, whatever you're working on





TO LEARN MORE ABOUT ALLINEA PRODUCTS

Attend Allinea technical tutorial & presentation !

Technical tutorial :Room PA2 "Gregory"Presentation :Amphi "Becquerel"

Friday 11h00-12h30 Friday 13h15-13h45

Thank you

Your contacts :

- Technical Support team :
- Sales team :

support@allinea.com sales@allinea.com

www.allinea.com