

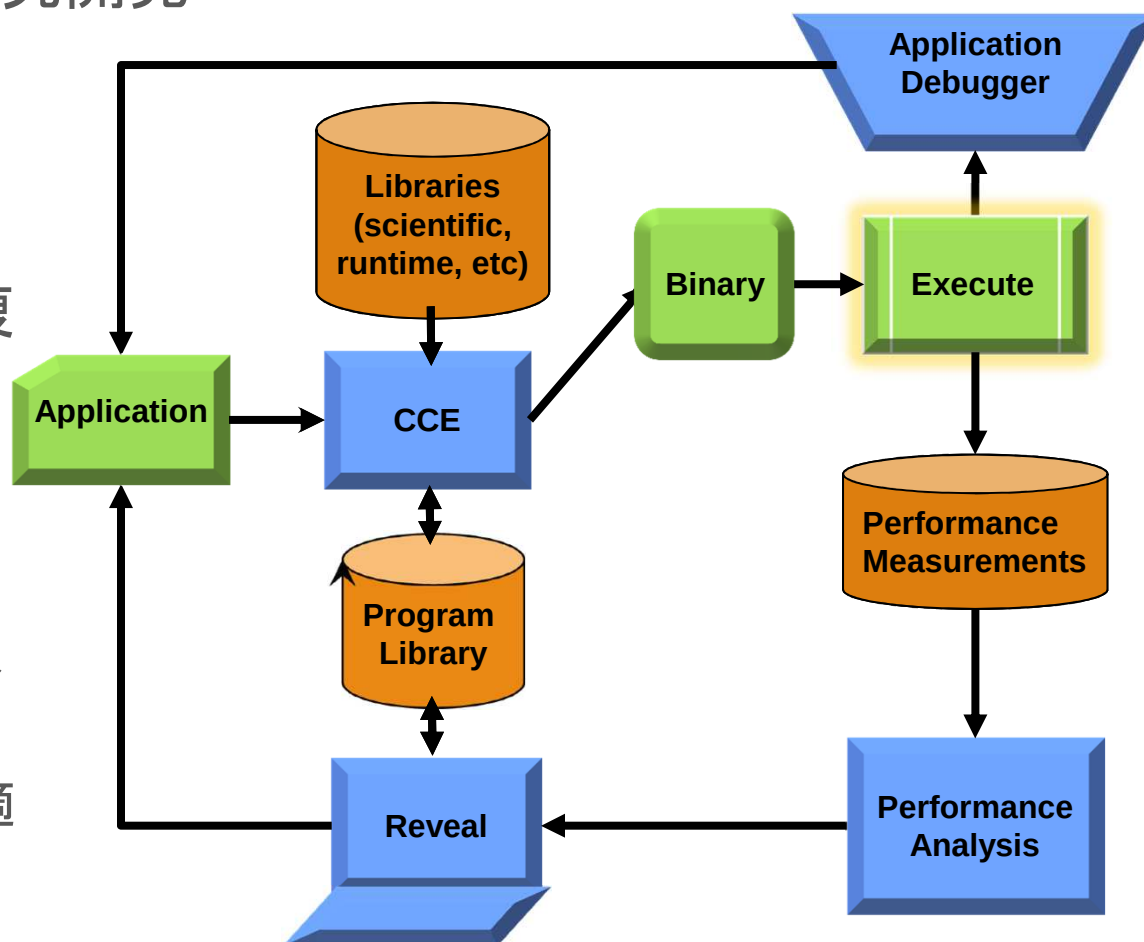


Crayプログラミング環境の開発と現状

寺西慶太
Cray Inc.

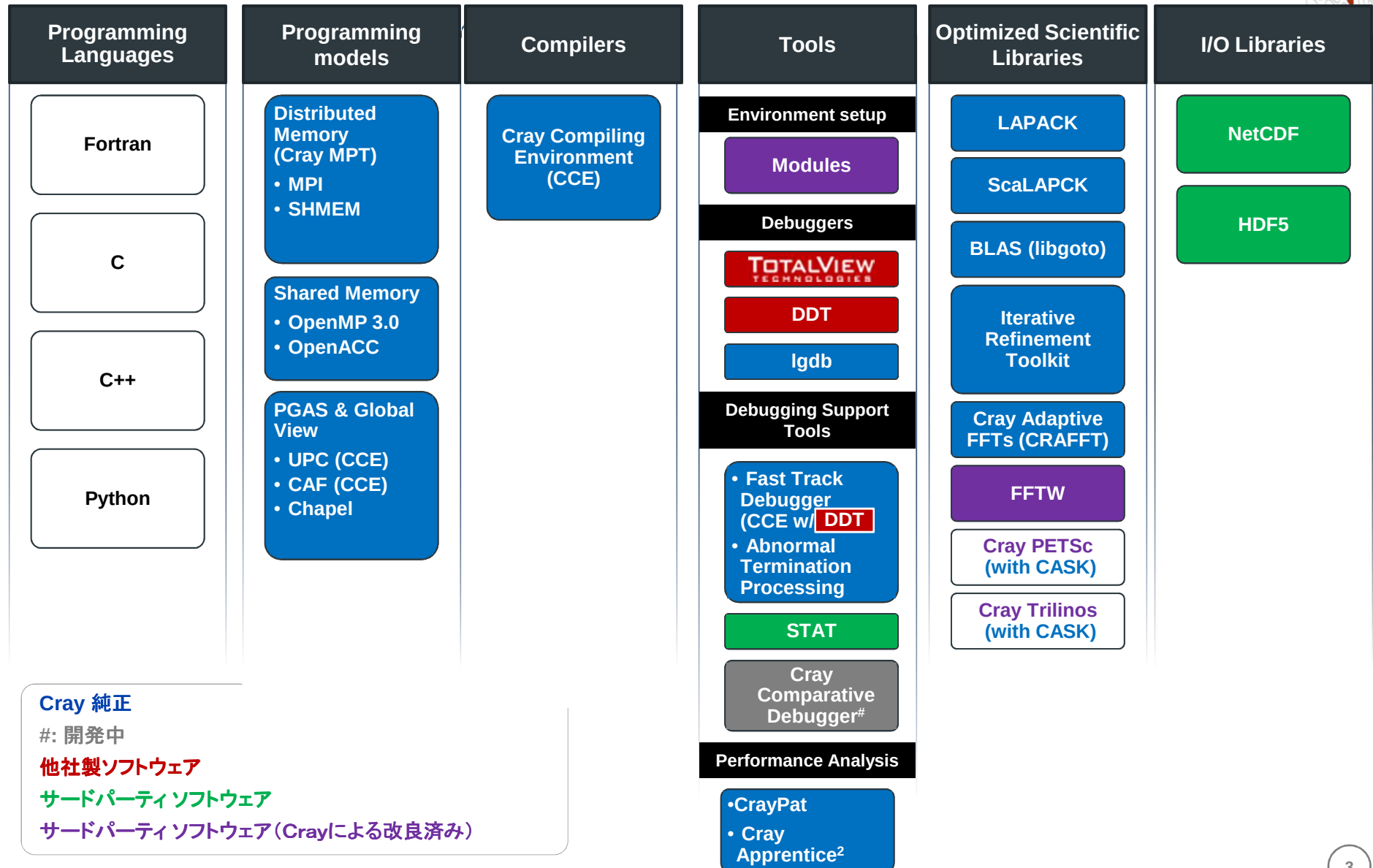
Crayのプログラミング環境へのビジョン

- Crayは**アプリケーション性能を最大限に向上**させることを目標にプログラミング環境を研究開発
- コンパイラ、ライブラリ、ツールの統合されたプログラミング環境で、HPCプログラミングの複雑さを克服
 - スケーラビリティ
 - 機能の拡張と自動化による使いやすさの向上
 - インタラクティブなツールでソースコードの変更を実行性能にフィードバックし最適化を支援



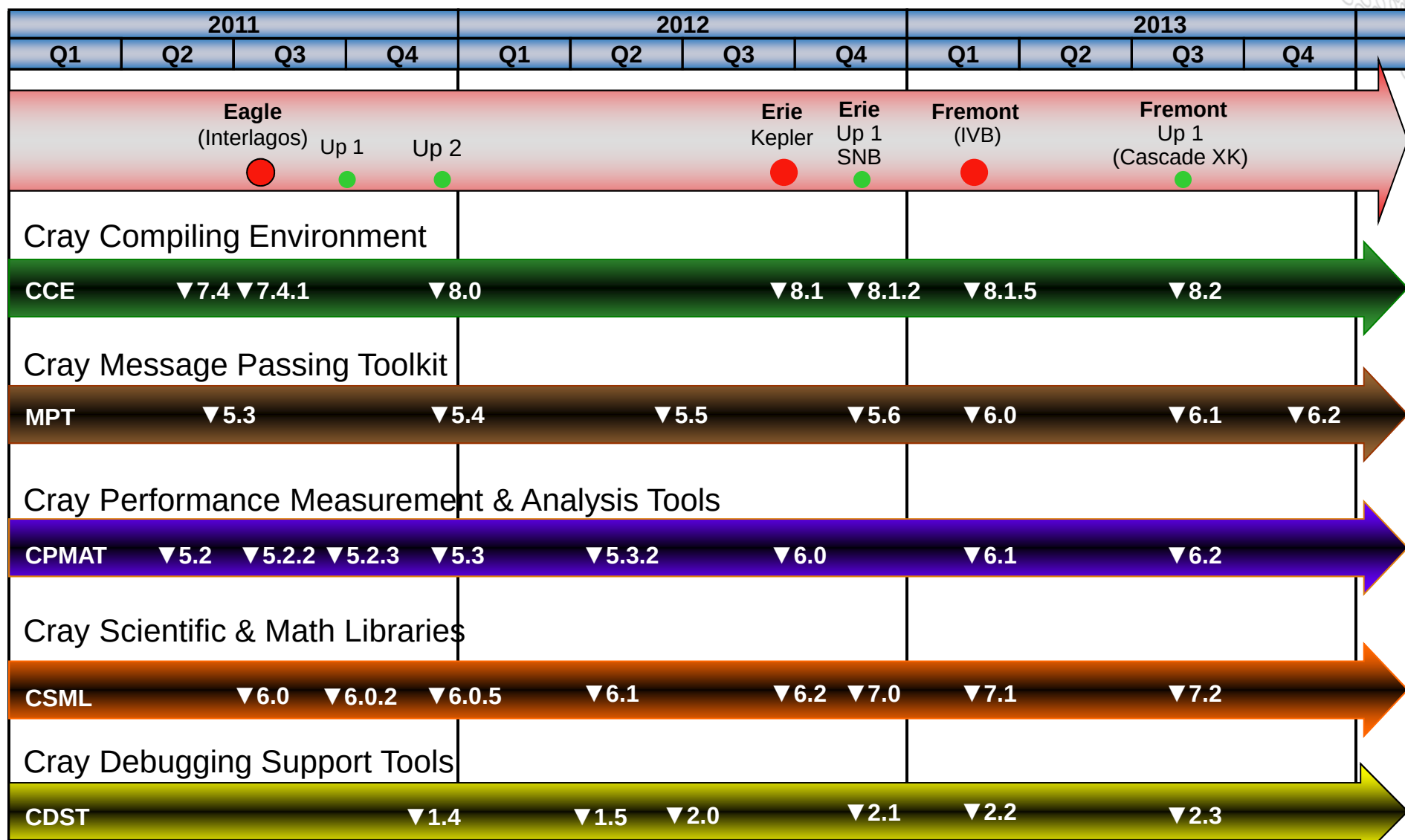


Cray のプログラミング環境





Cray Programming Environment Roadmap



CCE: Cray Compiling Environment



- 科学計算アプリケーションのコード最適化に特化
 - 自動ベクトル化
 - 自動共有メモリ並列化
- 標準化規格の準拠
 - **Fortran 2008 規格**
 - CCE 8.1から準拠の予定 (3Q12)
 - C++98/2003 規格準拠
 - **OpenMP 3.0 準拠**
 - OpenMP 3.1 and OpenMP 4.0規格にむけて積極的な活動
- OpenMP と自動共有メモリ並列化の統合
 - 同じランタイムライブラリに基づき実装され、スレッドプールを共有
 - OpenMP領域内外で更にループの再構築、スカラ命令の最適化
 - 自動スレッド並列化とOpenMPは共通の内部APIにアクセス
- PGAS 言語(UPC & Fortran Coarrays) の完全なサポートと実装の最適化
 - UPC 1.2 and Fortran 2008 coarray のサポート
 - 使用の際、プリプロセッサをコードに書き込む必要なし
 - Crayのネットワークハードウェアに合わせて実装
 - Allinea's DDTの完全サポートで、デバッグも容易に



Cray MPI と Cray SHMEM

● MPI

- アルゴンヌ研究所のMPICH2がベース
- 片方向通信、RMAの完全なサポート
 - 計算と通信のオーバラップ
- MPI-2 機能の完全サポート
 - MPI_Comm_spawnは除く
- MPI3 Forumで積極的な活動

● Cray SHMEM

- 最適化されたCray SHMEM ライブラリ
 - CrayT3Eの実装、デザインに基づく
 - Cray XE ではDistributed Memory Applications API (DMAPP) の上に実装
- 最近の新機能、性能強化:
 - ノード内ではプロセス間メモリコピーで通信
 - Cross Process Memory Mapping (XPMEM)
 - XPMEMで他のプロセスと自プロセス間のアドレス空間をマッピング
 - 分散メモリ版ロック
 - コレクティブ通信

Cray Performance Tools



- アプリケーションの性能データを性能解析、最適化へ導くツール群
 - CCEが出力する中間表現、最適化の情報を活用
- 使いやすさ、自動化の向上
 - GUI
 - 性能解析結果をプログラム実行へのフィードバック
- 複数のプログラミングモデルに対応
 - MPI, PGAS, OpenMP, OpenACC, SHMEM
- スケーラビリティの強化
 - 多ノードへの対応
 - レスポンスの向上
- 新機種への対応
 - Intel CPU
 - Aries Interconnect

Cray Performance Toolsのフィードバックによる性能チューニング例



MPI ランクの並びかえ:

- MPI通信はノード内では共有メモリコピーで実装
 - ノード間通信より大幅に高速
- 並列プログラム全体の通信の仕方、ロードバランスによってはノード内通信をより効果的に利用することができる
- MPIランクの並び替えをすることで**MPIの実行時間を大幅に下げる事が可能**
 - 最高で52%の事例も
- Cray Performance Toolでは性能結果を元にMPICH_RANK_ORDER.Gridというファイルが生成され、それをバッチファイルとして使う



ツールが推奨するMPIランク

```
# The 'USER Time hybrid'
rank order in this file
targets nodes with multi-
core
# processors, based on
Sent Msg Total Bytes
collected for:
#
# Program:
/lus/nid00023/malice/crayp
at/WORKSHOP/bh2o-
demo/Rank/sweep3d/src/swee
p3d
# Ap2 File:
sweep3d.gmpi-u.ap2
# Number PEs: 768
# Max PEs/Node: 16
#
# To use this file, make a
copy named
MPICH_RANK_ORDER, and set
the
# environment variable
MPICH_RANK_REORDER_METHOD
to 3 prior to
# executing the program.
#
0, 532, 64, 564, 32, 572, 96, 540
8, 596, 72, 524, 40, 604, 24, 58
8
104, 556, 16, 628, 80, 636, 56, 6
20, 48, 516, 112, 580, 88, 548, 1
20, 612
1, 403, 65, 435, 33, 411, 97, 443
9, 467, 25, 499, 105, 507, 41, 4
75
73, 395, 81, 427, 57, 459, 17, 41
9, 113, 491, 49, 387, 89, 451, 12
1, 483
6, 436, 102, 468, 70, 404, 38, 41
2, 14, 444, 46, 476, 110, 508, 78
, 500
86, 396, 30, 428, 62, 460, 54, 49
2, 118, 420, 22, 452, 94, 388, 12
6, 484
129, 563, 193, 531, 161, 571, 22
5, 539, 241, 595, 233, 523, 249,
603, 185, 555
153, 587, 169, 627, 137, 635, 20
1, 619, 177, 515, 145, 579, 209,
547, 217, 611
7, 405, 71, 469, 39, 437, 103, 41
3, 47, 445, 15, 509, 79, 477, 31,
501
111, 397, 63, 461, 55, 429, 87, 4
21, 23, 493, 119, 389, 95, 453, 1
27, 485
134, 402, 198, 434, 166, 410, 23
0, 442, 238, 466, 174, 506, 158,
394, 246, 474
190, 498, 254, 426, 142, 458, 15
0, 386, 182, 418, 206, 490, 214,
450, 222, 482
128, 533, 192, 541, 160, 565, 23
2, 525, 224, 573, 240, 597, 184,
557, 248, 605
168, 589, 200, 517, 152, 629, 13
6, 549, 176, 637, 144, 621, 208,
581, 216, 613
5, 439, 37, 407, 69, 447, 101, 41
5, 13, 471, 45, 503, 29, 479, 77,
511
53, 399, 85, 431, 21, 463, 61, 39
1, 109, 423, 93, 455, 117, 495, 1
25, 487
2, 530, 34, 562, 66, 538, 98, 522
10, 570, 42, 554, 26, 594, 50, 6
62
18, 514, 74, 586, 58, 626, 82, 54
6, 106, 634, 90, 578, 114, 618, 1
22, 610
135, 315, 167, 339, 199, 347, 25
9, 307, 231, 371, 239, 379, 191,
331, 247, 299
175, 363, 159, 323, 143, 355, 25
5, 291, 207, 275, 183, 283, 151,
267, 215, 223
133, 406, 197, 438, 165, 470, 22
9, 414, 245, 446, 141, 478, 237,
502, 253, 398
157, 510, 189, 462, 173, 430, 20
5, 390, 149, 422, 213, 454, 181,
494, 221, 486
130, 316, 260, 340, 194, 372, 16
2, 348, 226, 308, 234, 380, 242,
332, 250, 300
202, 364, 186, 324, 154, 356, 13
8, 292, 170, 276, 178, 284, 210,
218, 268, 146
4, 535, 36, 543, 68, 567, 100, 52
7, 12, 599, 44, 575, 28, 559, 76,
607
52, 591, 20, 631, 60, 639, 84, 51
9, 108, 623, 92, 551, 116, 583, 1
24, 615
3, 440, 35, 432, 67, 400, 99, 408
11, 464, 43, 496, 27, 472, 51, 5
04
19, 392, 75, 424, 59, 456, 83, 38
4, 107, 416, 91, 488, 115, 448, 1
23, 480
132, 401, 196, 441, 164, 409, 22
8, 433, 236, 465, 204, 473, 244,
393, 188, 497
252, 505, 140, 425, 212, 457, 15
6, 385, 172, 417, 180, 449, 148,
489, 220, 481
131, 534, 195, 542, 163, 566, 22
7, 526, 235, 574, 203, 598, 243,
558, 187, 606
251, 590, 211, 630, 179, 638, 13
9, 622, 155, 550, 171, 518, 219,
582, 147, 614
761, 660, 737, 652, 705, 668, 74
5, 692, 673, 700, 641, 684, 713,
644, 753, 724
729, 732, 681, 756, 721, 716, 76
4, 676, 697, 748, 689, 657, 740,
665, 649, 708
760, 528, 736, 536, 704, 560, 74
4, 520, 672, 568, 712, 592, 752,
552, 640, 600
728, 584, 680, 624, 720, 512, 69
6, 632, 688, 616, 664, 544, 608,
656, 648, 576
762, 659, 738, 651, 706, 667, 74
6, 643, 714, 691, 674, 699, 754,
683, 730, 723
722, 731, 763, 658, 642, 755, 73
9, 675, 707, 650, 682, 715, 698,
666, 690, 747
257, 345, 265, 313, 281, 305, 27
3, 337, 609, 369, 577, 377, 617,
329, 513, 529
545, 297, 633, 361, 625, 321, 58
5, 537, 601, 289, 553, 353, 593,
521, 569, 561
256, 373, 261, 341, 264, 349, 28
0, 317, 272, 381, 269, 309, 285,
333, 277, 365
352, 301, 320, 325, 288, 357, 32
8, 304, 360, 312, 376, 293, 296,
368, 336, 344
258, 338, 266, 346, 282, 314, 27
4, 370, 766, 306, 710, 378, 742,
330, 678, 362
646, 298, 750, 322, 718, 354, 75
8, 290, 734, 662, 686, 670, 726,
702, 694, 654
262, 375, 263, 343, 270, 311, 27
1, 351, 286, 319, 278, 342, 287,
350, 279, 374
294, 318, 358, 383, 359, 310, 29
5, 382, 326, 303, 327, 367, 366,
335, 302, 334
765, 661, 709, 663, 741, 653, 71
1, 669, 767, 655, 743, 671, 749,
695, 679, 703
677, 727, 751, 693, 647, 701, 71
7, 687, 757, 685, 733, 725, 719,
735, 645, 759
```

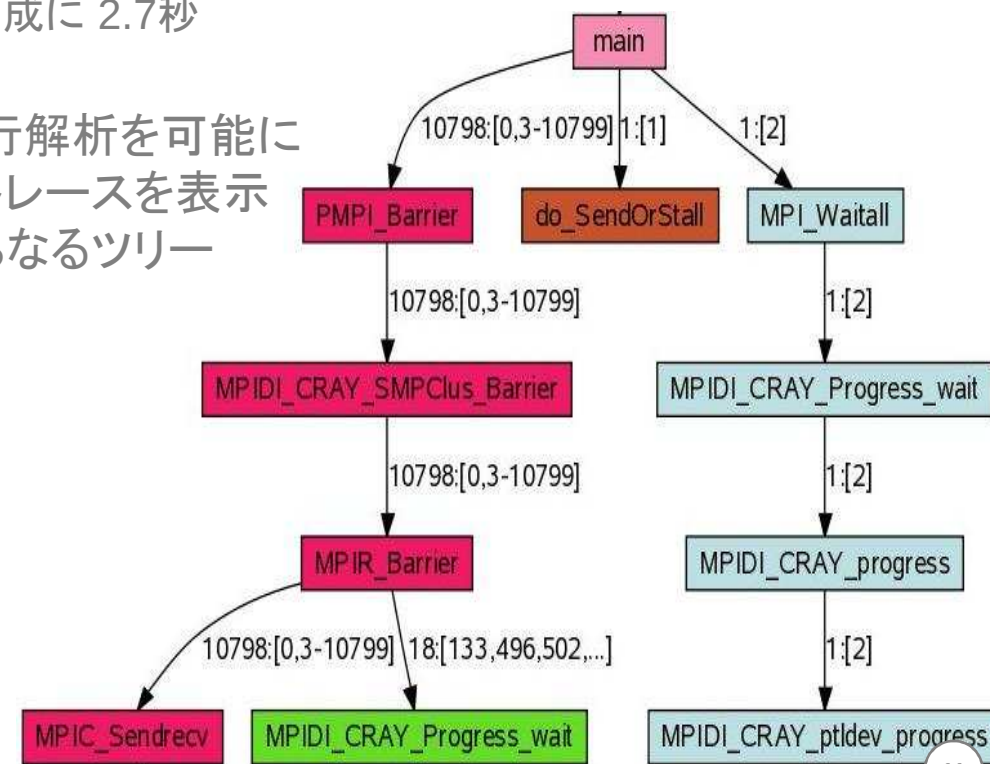


次世代デバugga

- 多数のプロセス、スレッドに対応できるデバugga
 - 最新の技術でスケラビリティー、生産性の向上
 - Wisconsin大で開発されたMRNetをインフラとして活用
 - STAT - Stack Trace Analysis Tool**
 - 統合された、バックトレースツリーの生成
 - 216,000MPIプロセスまで対応
 - ATP - Abnormal Termination Processing**
 - バグのあるプログラムの実行経路をツリー表示
 - Coreファイルの統合、縮小でスケラブルに実行.
 - Fast Track Debugging
 - 最適化されたコードのデバック
 - デバuggしたい箇所だけ、シンボル付きオブジェクトで実行
 - それ以外は最適化されたままで実行
 - アプリケーション実行そのままの環境でデバuggが可能に
 - Allinea's DDT 2.6 以降(2010年6月)
 - 従来のデバuggへの対応
 - TotalView, DDT, and gdb

Stack Trace Analysis Tool (STAT)

- 大規模アプリケーション向けスタックトレース
 - スタックトレース
 - スタックのバックトレースを1つのツリーを高速に生成
 - アプリケーション実行の全体像を可視化
 - 同じような実行経路をもつプロセスのスタックトレースの統合
 - SIMDプログラムの特徴
 - デバッグすべきプロセスを最小限に
 - 128,000MPIプロセスのトレース生成に 2.7秒
 - トレースの集約、解析
 - スケーラブルなアプリケーション実行解析を可能に
 - プログラムの経緯に従って複数のトレースを表示
 - 関数、サブルーチンの呼び出しからなるツリー





Automatic Termination Processing (ATP)

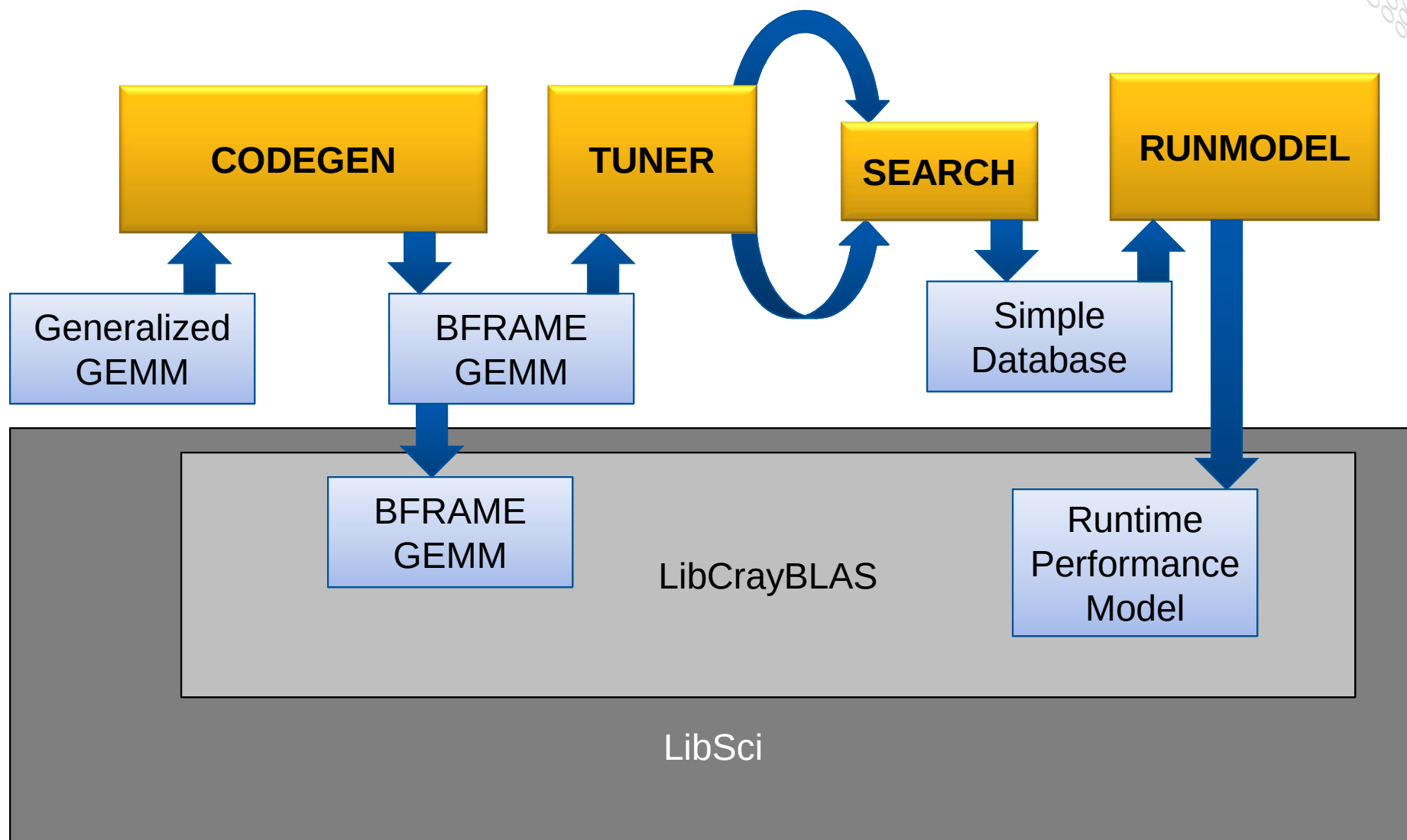
- 計算ノードに常駐してプロセスの異常終了を監視
 - MRNet上で実装
 - aprunでプログラム実行時に自動的に動作
 - 環境変数ATP_ENABLEDで、オンオフの切り替え
- アプリケーションの 異常動作に即座に反応
 - 最初に異常動作をしたプロセスのバックトレースをstderrに出力
 - そのプロセスのcoredumpの生成 (シェル環境でcoreサイズの制限がない場合)
 - StackwalkerAPI を用いて各プロセスのスタックバックトレースを収集
 - 最適化されたコード、オブジェクトに対しても実行される
- STAT (Stack Trace Analysis)同様な ツリー形式でバックトレースを表示
 - STATほど正確ではないが、異常終了する関数、サブルーチンをできるだけ早く発見できる
 - ツリーの末端に相当するノードのcoreファイル进行操作
 - もしくは、これらのノードをデバッグ実行



Adaptive Scientific Libraries

- Crayの科学計算ライブラリ
 - 標準API
 - 自動チューニング
 - 自動適応ライブラリ
- Cray **adaptive** model
 - ランタイム時に**ベスト**のカーネル、ライブラリを自動選択して実行
 - 開発過程で、膨大な量の性能解析を行い、その結果をコンパクトな形でライブラリに組み込むことで、自動選択のオーバーヘッドを最小限に抑えることが可能に
 - ライブラリ関数実行時に、入力パラメータを元にパフォーマンステーブルをルックアップ
 - 各々の問題サイズに最適化されたカーネルを選択

CrayのBLASチューニングの作業フロー





Crayの科学計算ライブラリの特長

● CASK (Cray Adaptive Sparse Kernels)

- Crayの自動チューニング技術を使って開発された疎行列ベクトル積カーネル
- PETSC, Trilinosの疎行列カーネルの性能の向上
- ユーザ側でコードの書き換えは不要
- 多種多様な非ゼロ分布で高性能を発揮
- 不完全LU, 不完全コレスキー前処理の性能の向上
- ベクトルが複数のケースにも対応
 - 疎行列固有値ソルバ
 - Uncertainty Quantification

● ScaLAPACK

- Geminiインターコネクト向けのチューニング

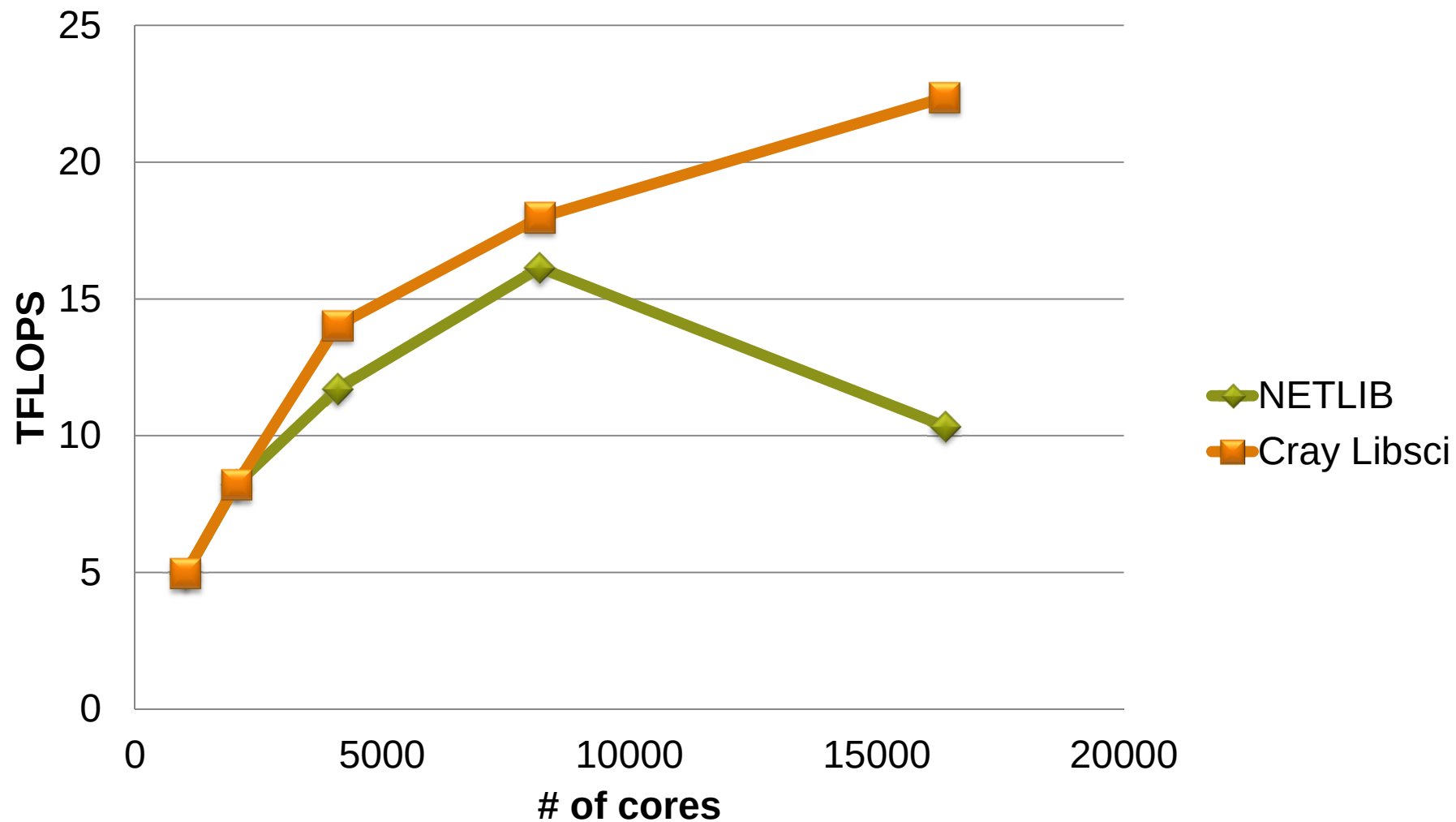
● FFTW

- ILプロセッサ向けメモリコピー性能の強化
- 544x544 2DFFT ノードあたり性能 (16PE、2スレッド)
- 最新版: 1.537GFLOPS
- 従来版: 1.063GFLOPS

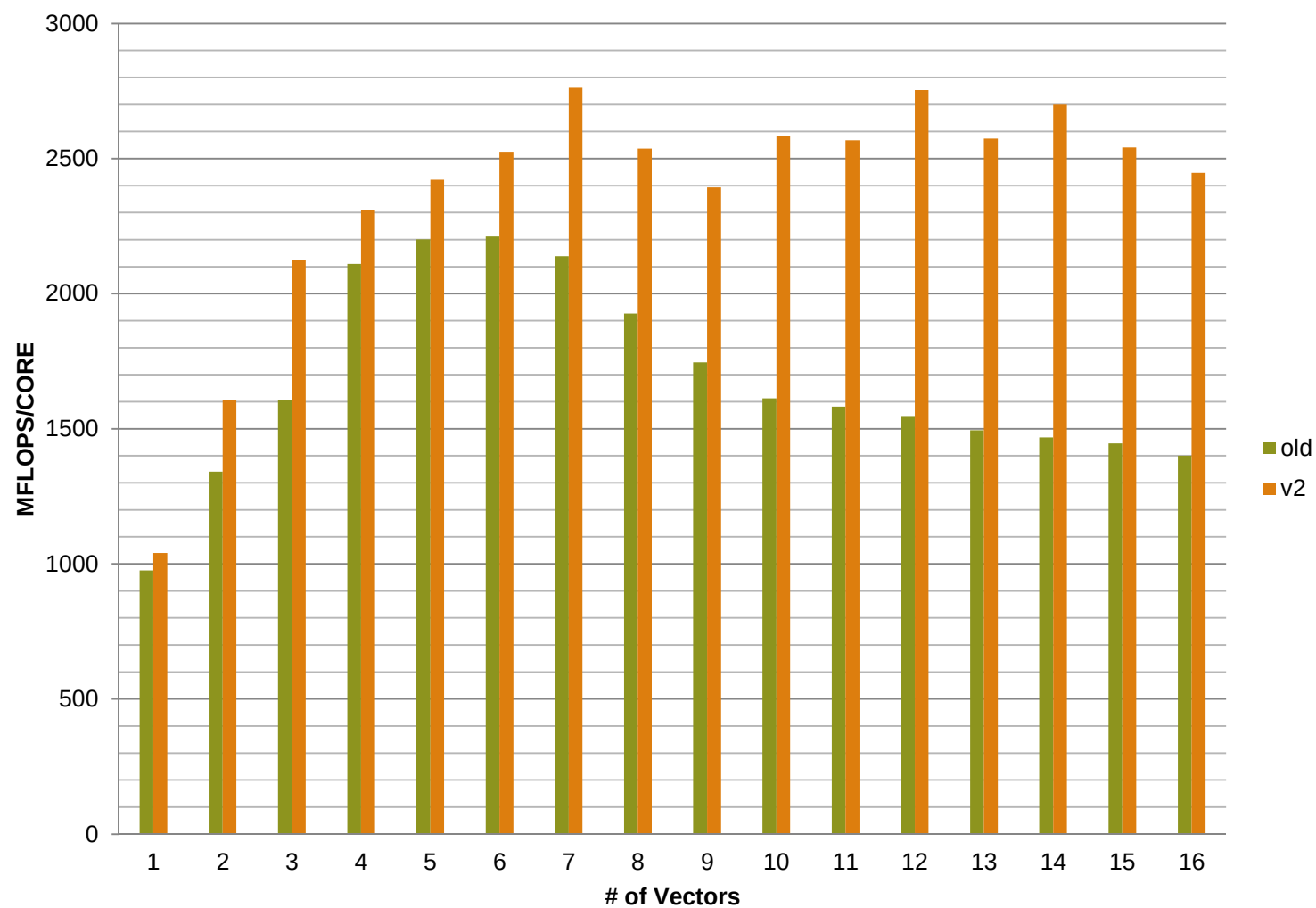
ScaLAPACKの性能

CRAY

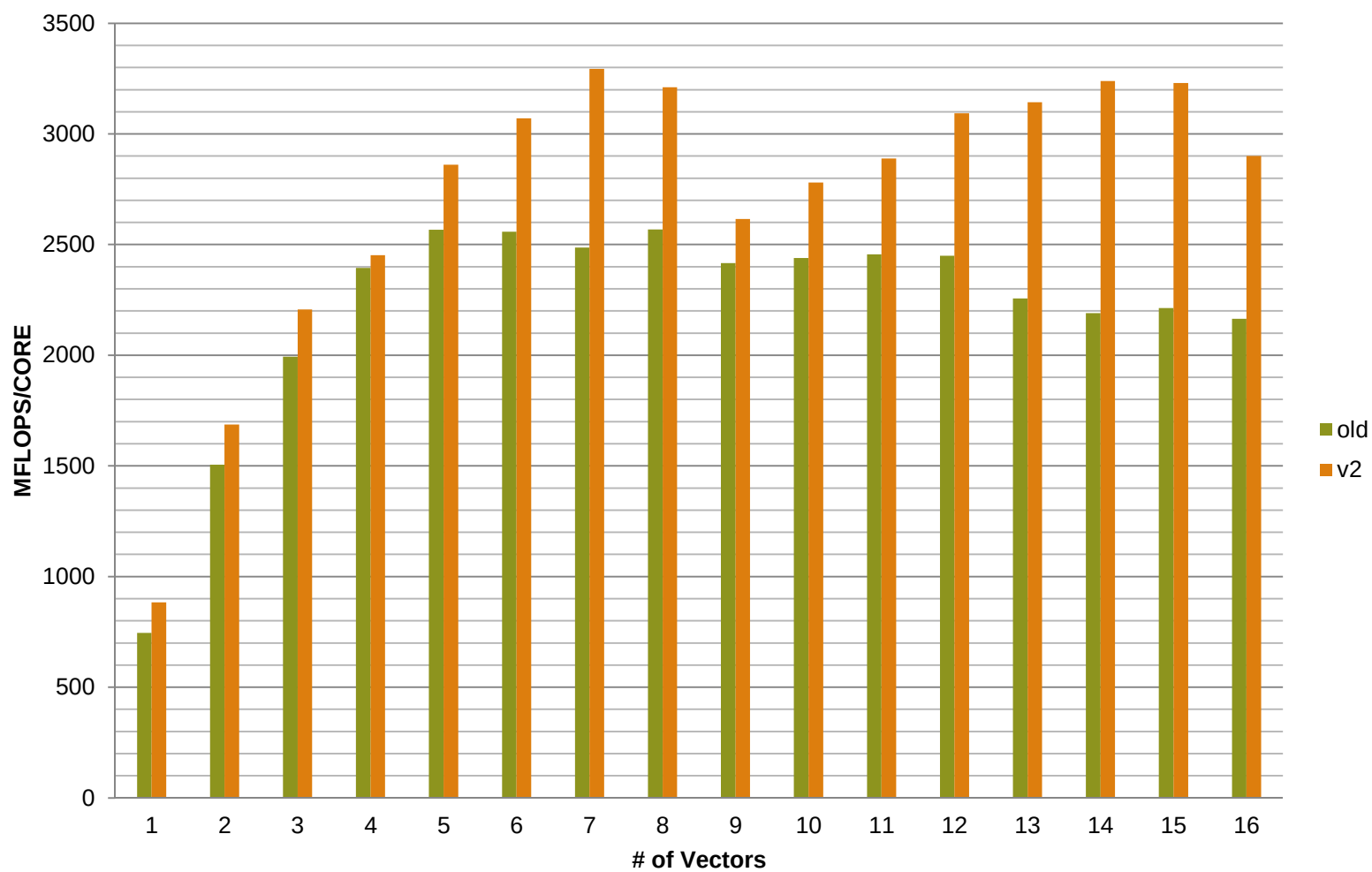
ScaLAPACK LU factorization on XE (M=131,072)



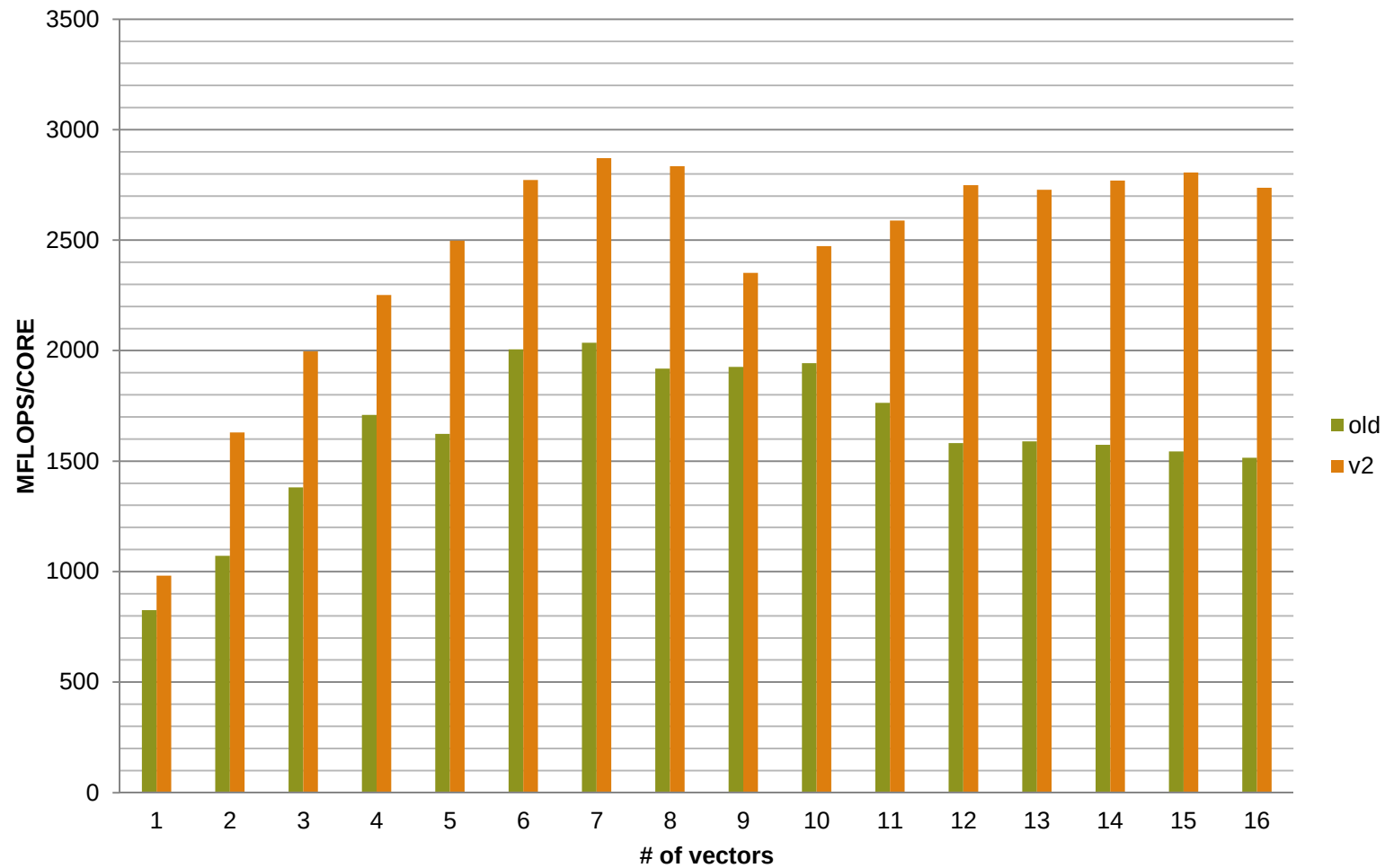
CASKの性能: CRYSTM01 matrix



CASKの性能: BCSSTK35



CASKの性能: AF23560



Cray Revealの機能

CRAY



Reveal

最適化のためのコードの書き換え、性能解析支援

クレイの既存の性能ツールとCCEのライブラリ関数を用いて、コンパイル時、ランタイム時の性能解析とデータを可視化

ソースコードと性能解析のデータを直接対応を可能に。ユーザはコードのどこを最適化、書き換えすべきかを容易に知ることができる

主な機能

ソースコードにコンパイラの最適化情報を注釈

各ループの最適化の情報
依存性などの情報を表示し、最適化が困難なケースをユーザに伝える

スコーピング解析

- 配列が共有、プライベート、曖昧であるかどうかを判別
- ユーザはその情報を元に、曖昧な配列のプライベート化を行う
- ユーザが直接コードを書き換えて、コンパイラの依存性解析の結果を無視して最適化

ソースコードの閲覧

CrayPatの結果を元にソースコードの各部分の性能情報を一緒に表示

ツールを使ってのノード内並列化の作業フロー

更に並列化が可能なソースの部分を探す

- X86システム上MPIプログラムが正しく動作することが前提
- CCEの自動スレッド化を試してみる
 - コンパイラがスレッド化可能なループを検知
- 計算量の多いループを探す
 - PerftoolsとCCEの両方を使うとループ毎の実行時間が分かる

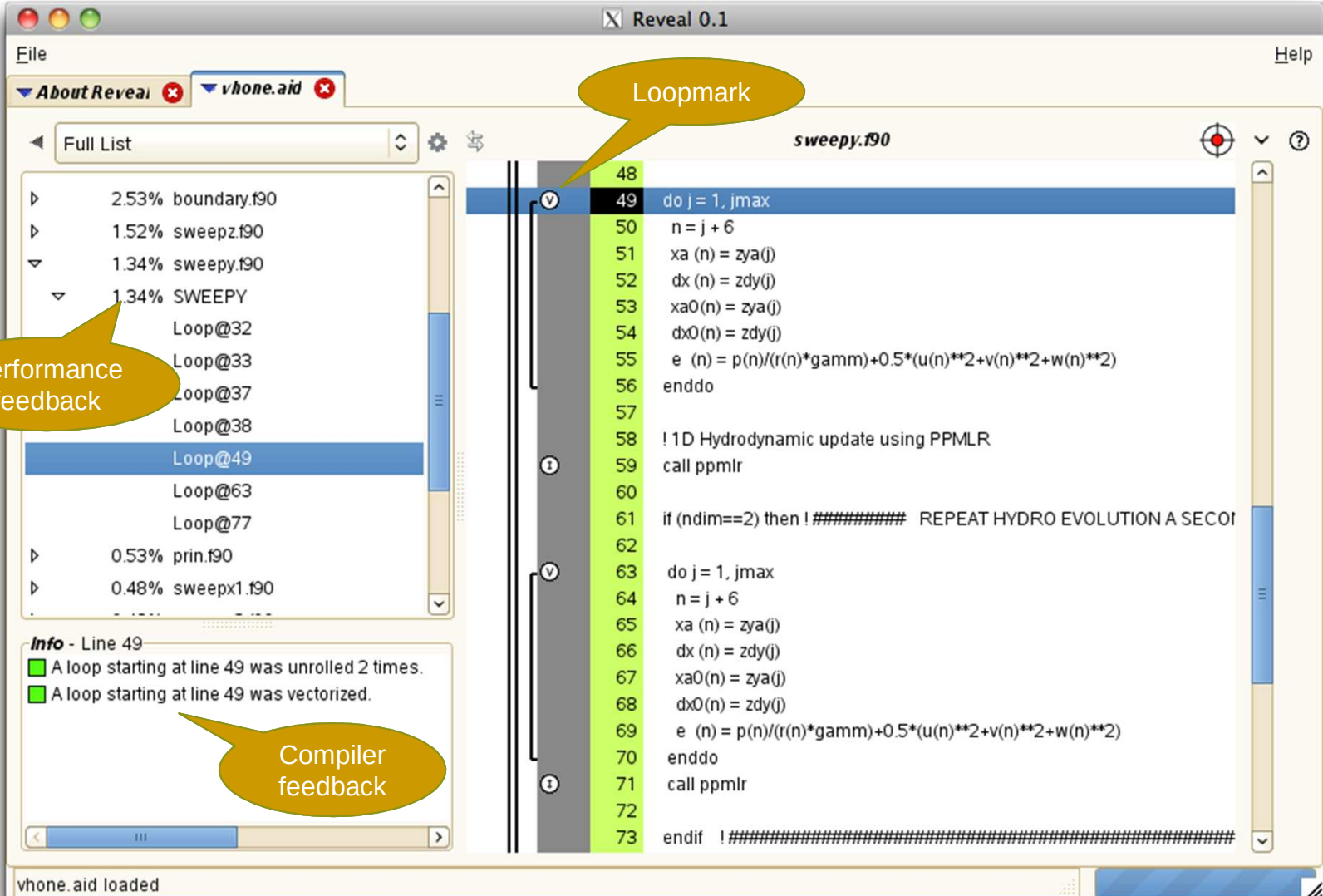
ループ内の計算を複数のスレッドに配分

- ループの並列化解析と再構築
- RevealとCCEを使うことで、各ループの情報(性能、最適化手法)とそれに対応するソースコードをGUI環境で操作

並列化ディレクティブの追加、アクセラレータ化

- OpenMPディレクティブを挿入
 - Revealのスコوپング機能
- X86システム上で動作の確認、性能のチェック
- OpenMPディレクティブをOpenACCディレクティブに書き換え

CCEによって生成されるループ情報の可視化



The screenshot shows the Reveal 0.1 application interface. On the left, a 'Full List' pane displays a hierarchy of code components with their respective percentages: boundary.f90 (2.53%), sweepz.f90 (1.52%), sweepy.f90 (1.34%), and SWEEPY (1.34%). Under SWEEPY, several loops are listed, with 'Loop@49' highlighted. Below this, an 'Info - Line 49' pane provides compiler feedback: 'A loop starting at line 49 was unrolled 2 times.' and 'A loop starting at line 49 was vectorized.' A yellow callout bubble labeled 'Performance feedback' points to the 'Loop@49' entry. Another yellow callout bubble labeled 'Compiler feedback' points to the 'Info' pane.

The main window displays the source code for 'sweepy.f90'. A yellow callout bubble labeled 'Loopmark' points to line 49, which is marked with a 'V' in a circle. The code snippet is as follows:

```

48
49 do j = 1, jmax
50   n = j + 6
51   xa (n) = zya(j)
52   dx (n) = zdy(j)
53   xa0(n) = zya(j)
54   dx0(n) = zdy(j)
55   e (n) = p(n)/(r(n)*gamm)+0.5*(u(n)**2+v(n)**2+w(n)**2)
56 enddo
57
58 ! 1D Hydrodynamic update using PPMLR
59 call ppmlr
60
61 if (ndim==2) then ! ##### REPEAT HYDRO EVOLUTION A SECO
62
63   do j = 1, jmax
64     n = j + 6
65     xa (n) = zya(j)
66     dx (n) = zdy(j)
67     xa0(n) = zya(j)
68     dx0(n) = zdy(j)
69     e (n) = p(n)/(r(n)*gamm)+0.5*(u(n)**2+v(n)**2+w(n)**2)
70   enddo
71   call ppmlr
72
73 endif ! #####
  
```

At the bottom of the window, a status bar indicates 'vhone.aid loaded'.

CCEによって生成されるループ情報の可視化



The screenshot displays the Reveal 0.1 application window. On the left, a file tree shows a project named 'vhone.aid' with a subdirectory 'SWEEPX1' containing several loops. The 'Loop@32' is selected. The main window shows a code listing for 'sweepx1.f90' starting at line 31. Line 32 is highlighted with a blue bar and a circled 'U' icon, indicating an unrolled loop. Below the code listing, an 'Info - Line 32' panel shows two green squares with the text 'A loop starting at line 32 w'. An orange callout bubble points to this panel with the text 'Integrated message 'explain support''. On the right, an 'Explain' dialog box is open, displaying the following text:

OPT_INFO: A loop starting at line %s was unrolled.

The compiler unrolled the loop. Unrolling creates a number of copies of the loop body. When unrolling an outer loop, the compiler attempts to fuse replicated inner loops - a transformation known as unroll-and-jam. The compiler will always employ the unroll-and-jam mode when unrolling an outer loop; literal outer loop unrolling may occur when unrolling to satisfy a user directive (pragma).

This message indicates that unroll-and-jam was performed with respect to the identified loop. A different message is issued when literal outer loop unrolling is performed, as this transformation is far less likely to be beneficial.

For sake of illustration, the following contrasts unroll-and-jam with literal outer loop unrolling.

```
# 434 "ptmp/pdgcgs/pdgcgs.tbs 81/bld_dir/build 64.ndb/pdgcgs/pdgcgs_ftn.msg.c"
DO J = 1,10
DO I = 1,100
A(I,J) = B(I,J) + 42.0
ENDDO
ENDDO

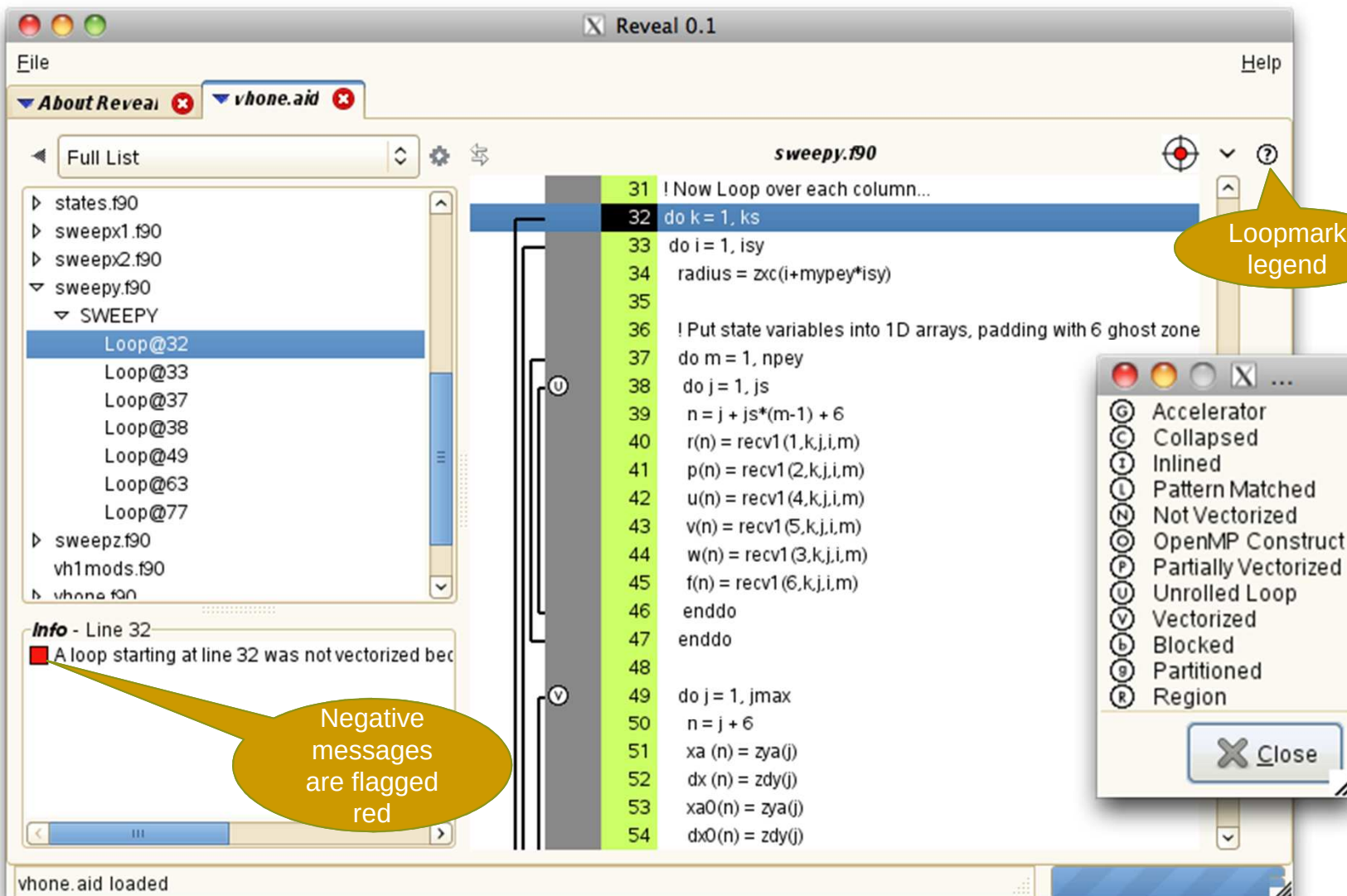
DO J = 1,10,2
DO I = 1,100
A(I,J) = B(I,J) + 42.0 ! unroll-and-jam
A(I,J+1) = B(I,J+1) + 42.0
ENDDO
ENDDO

DO J = 1,10,2
DO I = 1,100
A(I,J) = B(I,J) + 42.0 ! literal outer unroll
ENDDO
DO I = 1,100
A(I,J+1) = B(I,J+1) + 42.0
ENDDO
ENDDO
```

The literal outer unroll code performs the same sequence of memory operations as the original nest, while the unroll-and-jam transformation interleaves operations from outer loop iterations. The compiler employs literal outerloop unrolling only when the data dependencies in the loop, or a control flow impediment, prevent fusion of the replicated inner loops. Literal outer loop unrolling is generally not desirable. It is provided to ensure expected behavior and for those rare instances where the user has determined that it is beneficial.

At the bottom of the 'Explain' dialog, there are buttons for 'Explain other message...' and 'Close'.

CCEによって生成されるループ情報の可視化



The screenshot shows the Reveal 0.1 application window. The main window displays a code snippet from a file named `sweepy.f90`. The code is as follows:

```

31 ! Now Loop over each column...
32 do k = 1, ks
33   do i = 1, isy
34     radius = zxc(i+mypey*isy)
35   enddo
36   ! Put state variables into 1D arrays, padding with 6 ghost zone
37   do m = 1, npey
38     do j = 1, js
39       n = j + js*(m-1) + 6
40       r(n) = recv1(1,k,j,i,m)
41       p(n) = recv1(2,k,j,i,m)
42       u(n) = recv1(4,k,j,i,m)
43       v(n) = recv1(5,k,j,i,m)
44       w(n) = recv1(3,k,j,i,m)
45       f(n) = recv1(6,k,j,i,m)
46     enddo
47   enddo
48   do j = 1, jmax
49     n = j + 6
50     xa(n) = zya(j)
51     dx(n) = zdy(j)
52     xa0(n) = zya(j)
53     dx0(n) = zdy(j)
54   enddo

```

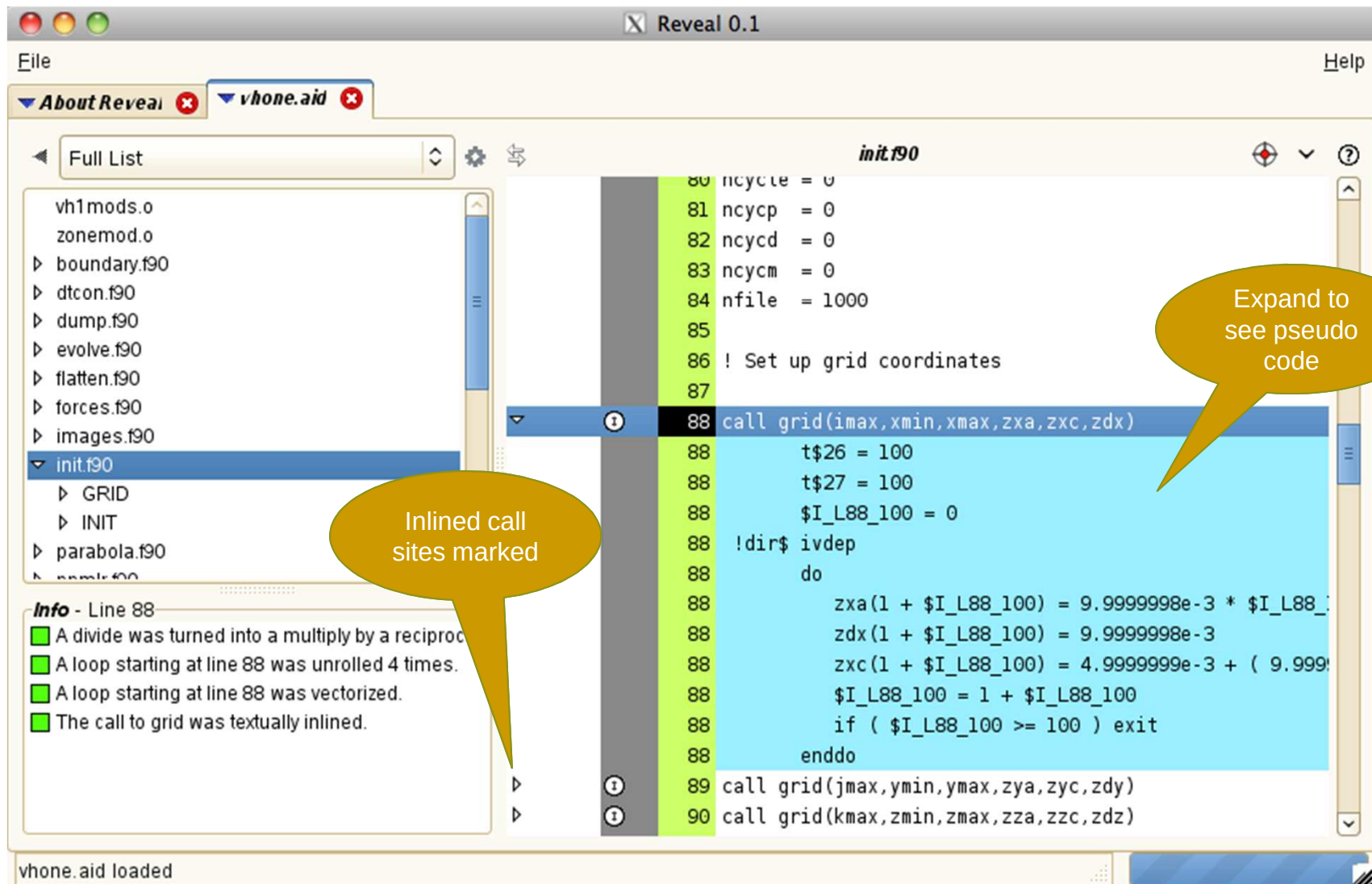
The code is annotated with loopmarks. A yellow callout bubble points to the loopmark legend, which is a small window with the following items:

- Accelerator
- Collapsed
- Inlined
- Pattern Matched
- Not Vectorized
- OpenMP Construct
- Partially Vectorized
- Unrolled Loop
- Vectorized
- Blocked
- Partitioned
- Region

A red square icon in the bottom left corner of the code editor indicates a negative message. A yellow callout bubble points to this icon with the text: "Negative messages are flagged red".

The status bar at the bottom of the window shows "vhone.aid loaded".

インライン化された関数の擬似コードの表示



Reveal 0.1

File Help

▼ About Reveal x ▼ vhone.aid x

Full List

- vh1mods.o
- zonemod.o
- ▷ boundary.f90
- ▷ dtcon.f90
- ▷ dump.f90
- ▷ evolve.f90
- ▷ flatten.f90
- ▷ forces.f90
- ▷ images.f90
- ▼ init.f90
 - ▷ GRID
 - ▷ INIT
 - ▷ parabola.f90
 - ▷ sample.f90

init.f90

```

80 ncycle = 0
81 ncycp = 0
82 ncycd = 0
83 ncycm = 0
84 nfile = 1000
85
86 ! Set up grid coordinates
87
88 call grid(imax,xmin,xmax,zxa,zxc,zdx)
88     t$26 = 100
88     t$27 = 100
88     $I_L88_100 = 0
88     !dir$ ivdep
88     do
88         zxa(1 + $I_L88_100) = 9.9999998e-3 * $I_L88_100
88         zdx(1 + $I_L88_100) = 9.9999998e-3
88         zxc(1 + $I_L88_100) = 4.9999999e-3 + ( 9.9999999e-3 * $I_L88_100 )
88         $I_L88_100 = 1 + $I_L88_100
88         if ( $I_L88_100 >= 100 ) exit
88     enddo
89 call grid(jmax,ymin,ymax,zya,zyc,zdy)
90 call grid(kmax,zmin,zmax,zza,zzc,zdz)
  
```

Expand to see pseudo code

Inlined call sites marked

Info - Line 88

- A divide was turned into a multiply by a reciprocal.
- A loop starting at line 88 was unrolled 4 times.
- A loop starting at line 88 was vectorized.
- The call to grid was textually inlined.

vhone.aid loaded

Revealによるスコーピング

Reveal 0.1

File Help

▼ About Reveal x ▼ vhone.aid x

Full List

- states.f90
- sweepx1.f90
- sweepx2.f90
- ▼ sweepy.f90
 - ▼ SWEEPY
 - Loop@32
 - Loop@33
 - Loop@37
 - Loop@38
 - Loop@49
 - Loop@63
 - Loop@77
 - sweepz.f90
 - vh1mods.f90
 - ▼ vhone.f90

Info - Line 32

■ A loop starting at line 32 was not vectorized because...

sweepy.f90

```
31 ! Now Loop over each column...
32 do k = 1, ks
33   do i = 1, isy
34     radius = zxc(i+mypey*isy)
35   enddo
36   ! Put state variables into 1D arrays, padding with 6 ghost zone
37   do m = 1, npey
38     do j = 1, js
39       n = j + js*(m-1) + 6
40       r(n) = recv1(1,k,j,i,m)
41       p(n) = recv1(2,k,j,i,m)
42       u(n) = recv1(4,k,j,i,m)
43       v(n) = recv1(5,k,j,i,m)
44       w(n) = recv1(3,k,j,i,m)
45       f(n) = recv1(6,k,j,i,m)
46     enddo
47   enddo
48
49   do j = 1, jmax
50     n = j + 6
51     xa(n) = zya(j)
52     dx(n) = zdy(j)
53     xa0(n) = zya(j)
54     dx0(n) = zdy(j)
55   enddo
56 enddo
```

Scope Loops

Reveal Scoping Tool

Line #	File or Source Line
0	/home/users/heidi/demo/sweepx2.f90
28	"do k = 1, ks"
0	/home/users/heidi/demo/sweepx1.f90
28	"do k = 1, ks"
0	/home/users/heidi/demo/sweepy.f90
32	"do k = 1, ks"
0	/home/users/heidi/demo/sweepz.f90
48	"do j = 1, js"

Start Cancel /home/users/heidi/demo/sweepy.f90 Close

Revealによるスコーピング



The screenshot displays the Reveal 0.1 application window. The left sidebar shows a file tree with 'riemann.f90' selected. The main pane shows the source code of 'riemann.f90'. A yellow callout bubble points to the 'Loop@28' entry in the file list, stating: "Loops with scoping information highlighted – red needs user assistance". Another yellow callout bubble points to the 'Unknown' scope value in the 'OpenMP Context' dialog, stating: "User scopes unknowns". The 'OpenMP Context' dialog is open, showing a table of variables and their scopes.

Name	Type	Scope	Info
radius	Scalar	Unknown	FAIL-Last defining iteration not known
a	Scalar	Private	
ai	Scalar	Private	
amid	Scalar	Private	
ar	Scalar	Private	
b	Scalar	Private	
bi	Scalar	Private	
c	Scalar	Private	
cdtdx	Scalar	Private	

Below the table, there are checkboxes for 'First/Last Private' and 'Enable First Private', and a 'Reduction' dropdown menu set to 'None'. At the bottom of the dialog are 'Dump Data' and 'Close' buttons.

Revealによるスコーピング

The screenshot shows the Reveal 0.1 application window. The main editor displays Fortran code from `sweepx1.f90`. The code includes a loop starting at line 28, which is highlighted in blue. The code is as follows:

```
27 ! Now Loop over each row...
28 do k = 1, ks
29   do j = 1, js
30
31     ! Put state variables into 1D arrays, padding with 6 ghost zone
32     do i = 1,imax
33       n = i + 6
34       r (n) = zro(i,j,k)
35       p (n) = zpr(i,j,k)
36       u (n) = zux(i,j,k)
37       v (n) = zuy(i,j,k)
38       w (n) = zuz(i,j,k)
39       f (n) = zfl(i,j,k)
40
41       xa0(n) = zxa(i)
42       dx0(n) = zdx(i)
43       xa (n) = zxa(i)
44       dx (n) = zdx(i)
45       p (n) = max(smallp,p(n))
46       e (n) = p(n)/(r(n)*gamm)+0.5*(u(n)**2+v(n)**2)
47     enddo
48
```

The left sidebar shows a file list with `sweepx1.f90` selected. The bottom status bar indicates `vhone.aid loaded`. A yellow callout bubble points to the `svel` variable in the code, with the text "Assist user with OpenMP hints". Another yellow callout bubble points to the `Construct` window, which displays a table of variables and their types and scopes.

Construct Window Table:

Name	Type	Scope	Info
xa2	Scalar	Private	
xa3	Scalar	Private	
zlf	Scalar	Private	
zrgh	Scalar	Private	
svel	Scalar	Shared	
dinflo	Scalar	Shared	
dofflo	Scalar	Shared	
dt	Scalar	Shared	
gam	Scalar	Shared	
gamm	Scalar	Shared	

The `Construct` window also includes checkboxes for `Enable First Private` and `Enable Last Private`, a `Reduction` dropdown set to `MAX`, and a `Search` field. Buttons for `Dump Data` and `Close` are at the bottom.

A yellow callout bubble at the bottom of the code editor contains the following text:

```
private (a,ai b,bi,c,...)
reduction (MAX:svel)
firstprivate (amid,ar,cdtdx,clft,...)
```

Cray のアクセラレータコンピューティングへのビジョン

The Cray logo is located in the top right corner of the slide. It consists of the word "CRAY" in a blue, sans-serif font. To the right of the text is a decorative graphic of a grid of circles, with some circles highlighted in red and yellow.

プログラミングの複雑さがアクセラレータコンピューティングへの障害である。

- 複数のプラットフォームで動く単一のプログラミングモデルが必須
 - ポータブルな 表現で各レベルの並列化が実装でき
 - プログラミングモデル、最適化手法がマルチコアx86CPUとあまり変わらない
 - ユーザは同じソースコードで各プラットフォームに合わせて実装ができる

Crayは統合されたプログラミング環境を、コンパイラ、ライブラリ、ツールによって提供し、高性能なアプリケーション開発を容易にすることを目標に研究開発

Crayの提供するプログラミング環境

- OpenACCディレクティブが実装されたFortran, C, C++ コンパイラ
 - ディレクティブによるアクセラレータプログラミングと最適化
- Crayコンパイラと統合された性能ツールとデバグガ
 - CUDAレベルでデバグ、コード性能解析をする必要がない
- アクセラレータ向け科学計算ライブラリ



XKシステムノード上でのプログラミング

Fortran, C, and C++ コンパイラ

OpenACC ディレクティブでプログラムを記述

- データ転送、ポインタの受け渡し等の記述が容易
- コンパイラがアクセラレータ、x86向け両方の最適化
- CUDAで書かれたカーネル、関数の組み込みも可能
- ノード並列デバッガ DDT、TotalViewの利用が可能



開発中のCray **Reveal**はコンパラが生成するソースコードの内部表記を元に性能解析、最適化の作業を支援

- GUIでソースコードを閲覧しながらループのGPU並列化、ベクトル化等を行える
 - スコーピングでコードの移植、最適化を支援
 - Crayの性能解析ツールの情報と組み合わせて、コードの最適化も可能

科学計算ライブラリ

- OpenACC、CUDAと互換
- 従来のAPIをそのまま継承
- Crayの自動チューニング技術

基本例題: リダクション

配列の総和を求める
Fortranだと4行

```
a=0.0  
  
do i = 1,n  
  a = a + b(i)  
end do
```


CUDAで書いたリダクションコード

```
__global__ void reduce0(int *g_odata, int
*g_odata)
{
extern __shared__ int sdata[];

unsigned int tid = threadIdx.x;
unsigned int i = blockIdx.x*blockDim.x +
threadIdx.x;
sdata[tid] = g_odata[i];
__syncthreads();

for(unsigned int s=1; s < blockDim.x; s *= 2) {
if ((tid % (2*s)) == 0) {
sdata[tid] += sdata[tid + s];
}
__syncthreads();
}

if (tid == 0) g_odata[blockIdx.x] = sdata[0];
}

extern "C" void reduce0_cuda_(int *n, int *a,
int *b)
{
int *b_d, red;
const int b_size = *n;

cudaMalloc((void **) &b_d , sizeof(int)*b_size);
cudaMemcpy(b_d, b, sizeof(int)*b_size,
cudaMemcpyHostToDevice);
```

```
dim3 dimBlock(128, 1, 1);
dim3 dimGrid(2048, 1, 1);
dim3 small_dimGrid(16, 1, 1);

int smemSize = 128 * sizeof(int);
int *buffer_d, *red_d;
int *small_buffer_d;

cudaMalloc((void **) &buffer_d ,
sizeof(int)*2048);
cudaMalloc((void **) &small_buffer_d ,
sizeof(int)*16);
cudaMalloc((void **) &red_d , sizeof(int));

reduce0<<< dimGrid, dimBlock, smemSize >>>(b_d,
buffer_d);

reduce0<<< small_dimGrid, dimBlock, smemSize
>>>(buffer_d, small_buffer_d);

reduce0<<< 1, 16, smemSize >>>(small_buffer_d,
red_d);

cudaMemcpy(&red, red_d, sizeof(int),
cudaMemcpyDeviceToHost);

*a = red;

cudaFree(buffer_d);
cudaFree(small_buffer_d);
cudaFree(b_d);
}
```

更に最適化されたリダクション

```
template<class T>
struct SharedMemory
{
    __device__ inline operator T*()
    {
        extern __shared__ int __smem[];
        return (T*)__smem;
    }

    __device__ inline operator const T*() const
    {
        extern __shared__ int __smem[];
        return (T*)__smem;
    }
};

template <class T, unsigned int blockSize, bool nlsPow2>
__global__ void
reduce6(T *g_idata, T *g_odata, unsigned int n)
{
    T *sdata = SharedMemory<T>();

    unsigned int tid = threadIdx.x;
    unsigned int i = blockIdx.x*blockSize*2 + threadIdx.x;
    unsigned int gridSize = blockSize*2*gridDim.x;

    T mySum = 0;
    while (i < n)
    {
        mySum += g_idata[i];
        if (nlsPow2 || i + blockSize < n)
            mySum += g_idata[i+blockSize];
        i += gridSize;
    }
    sdata[tid] = mySum;
    __syncthreads();

    if (blockSize >= 512) { if (tid < 256) { sdata[tid] = mySum = mySum
+ sdata[tid + 256]; } __syncthreads(); }
    if (blockSize >= 256) { if (tid < 128) { sdata[tid] = mySum = mySum
+ sdata[tid + 128]; } __syncthreads(); }
    if (blockSize >= 128) { if (tid < 64) { sdata[tid] = mySum = mySum
+ sdata[tid + 64]; } __syncthreads(); }
```

```
if (tid < 32)
{
    volatile T* smem = sdata;
    if (blockSize >= 64) { smem[tid] = mySum = mySum + smem[tid + 32]; }
    if (blockSize >= 32) { smem[tid] = mySum = mySum + smem[tid + 16]; }
    if (blockSize >= 16) { smem[tid] = mySum = mySum + smem[tid + 8]; }
    if (blockSize >= 8) { smem[tid] = mySum = mySum + smem[tid + 4]; }
    if (blockSize >= 4) { smem[tid] = mySum = mySum + smem[tid + 2]; }
    if (blockSize >= 2) { smem[tid] = mySum = mySum + smem[tid + 1]; }
}

if (tid == 0)
    g_odata[blockIdx.x] = sdata[0];
}
extern "C" void reduce6_cuda_(int *n, int *a, int *b)
{
    int *b_d;
    const int b_size = *n;

    cudaMalloc((void **) &b_d , sizeof(int)*b_size);
    cudaMemcpy(b_d, b, sizeof(int)*b_size, cudaMemcpyHostToDevice);

    dim3 dimBlock(128, 1, 1);
    dim3 dimGrid(128, 1, 1);
    dim3 small_dimGrid(1, 1, 1);
    int smemSize = 128 * sizeof(int);
    int *buffer_d;
    int small_buffer_d[4],*small_buffer_d;

    cudaMalloc((void **) &buffer_d , sizeof(int)*128);
    cudaMalloc((void **) &small_buffer_d , sizeof(int));
    reduce6<int,128,false><<< dimGrid, dimBlock, smemSize >>>(b_d,buffer_d,
b_size);
    reduce6<int,128,false><<< small_dimGrid, dimBlock, smemSize
>>>(buffer_d, small_buffer_d,128);
    cudaMemcpy(small_buffer, small_buffer_d, sizeof(int),
cudaMemcpyDeviceToHost);

    *a = *small_buffer;

    cudaFree(buffer_d);
    cudaFree(small_buffer_d);
    cudaFree(b_d);
}
```



OpenACCでリダクションを実装する場合

コンパイラが以下の機能を実行:

- !\$ACC内の並列化のできるループを確認
- カーネル化する必要があるか判断
- アクセラレータ向けコードCPU向けコードに分割
- ホスト側とアクセラレータ側で計算実行の分担
 - MIMD もしくはSIMDスタイルで実行
- データ転送
 - GPUメモリの割り当てと開放を!\$ACC領域の最初と最後で実行
 - CPUとGPUでデータ転送

```
!$acc data present(a,b)
!$acc parallel
```

```
a = 0.0
```

```
!$acc loop reduction(+:a)
```

```
do i = 1,n
  a = a + b(i)
end do
```

```
!$acc end parallel
!$acc end data
```



コンパイラからの実行オブジェクト以外の出力

```
90.    subroutine sum_of_int_4(n,a,b)
91.    use global_data
92.    integer*4 a,b(n)
93.    integer*8 start_clock, elapsed_clocks, end_clock
94.    !$acc data present(a,b)
95. G----< !$acc parallel
96. G    a = 0.0
97. G    !$acc loop reduction(+:a)
98. G g--< do i = 1,n
99. G g    a = a + b(i)
100. G g--> end do
101. G----> !$acc end parallel
102.    !$acc end data
103.    end subroutine sum_of_int_4
```

ftn-6413 ftn: ACCEL File = gpu_reduce_int_cce.F90, Line = 94
A data region was created at line 94 and ending at line 107.

ftn-6405 ftn: ACCEL File = gpu_reduce_int_cce.F90, Line = 95
A region starting at line 95 and ending at line 101 was placed on the accelerator.

ftn-6430 ftn: ACCEL File = gpu_reduce_int_cce.F90, Line = 98
A loop starting at line 98 was partitioned across the threadblocks and the 128 threads within a threadblock.



リダクションの性能

プログラム言語	実行元	コードの長さ	Gflops性能	X86 1コアに対する性能
Fortran	x86CPU1コア	4	2.0 Gflops	1.0
CUDA	GPU	30	1.74 Gflops	0.87
最適化版 CUDA	GPU	69	10.5 Gflops	5.25
OpenACC	GPU	9	8.32 Gflops	4.16

Cray Performance Tools for Accelerators

- スケーラビリティ
 - 多数のノードでも短いレスポンス時間
 - 性能結果は1ファイル、ディレクトリに集約
- アプリケーション全体の性能情報をユーザに
 - 性能データをソースコードにマッピング
 - 性能データをディレクティブ毎にグループ化
 - CPU側、アクセラレータ側の両方の性能解析が可能
- CPUとGPUの性能情報を一括に管理が可能
 - 性能情報
 - アクセラレータの実行時間、CPUの実行時間、CPUとアクセラレータ間のデータ転送の評価、解析
 - カーネル単位の性能データ
 - アクセラレータのハードウェアカウンタを利用



Performance Tools Example

```
t1 = gettime()
stream_counter = 1
DO j = 1,Nchunks
  my_stream = Streams(stream_counter)

  !$acc parallel loop
    DO i = 1,Nvec
      b(i,j) = SQRT(EXP(a(i,j)*2d0))
      b(i,j) = LOG(b(i,j)**2d0)/2d0
    ENDDO
  !$acc end parallel loop

  stream_counter = MOD(stream_counter,3) + 1
ENDDO
!$acc wait
t2 = gettime()
!$acc end data
```

ループのGPU化ディレクティブ

CPU,GPU間のデータ転送に関する記述が無いので、コンパイラが自動的に!\$acc data copyを挿入。
その結果GPU実行毎にa(),b()全体をCPU-GPU間でコピー、コピーバック

Performance Tools Example

```
ftn -rad -hnocaf -c -o toaa2.o toaa2.F90
ftn -rad -hnocaf -o toaa2.x toaa2.o
pat_build -w toaa2.x
aprun toaa2.x+pat
4999899.3359271679
Time = 88.750109565826278
Experiment data file written:
/lus/scratch/beyerj/openacc/toaa/toaa2.x+pat+10112-43t.xf
Application 1880125 resources: utime ~83s, stime ~7s
pat_report -T toaa2.x+pat+10112-43t.xf
```



Performance Tools Example

Table 1: Profile by Function Group and Function

Time%	Time	Imb. Time	Imb. Time%	Calls	Group Function
100.0%	88.902394	--	--	5003.0	Total
100.0%	88.902394	--	--	5003.0	USER
75.4%	67.041165	--	--	1000.0	toaa_.ACC_COPY@li.59
24.3%	21.629574	--	--	1000.0	toaa_.ACC_COPY@li.65
0.2%	0.155233	--	--	1.0	toaa_
0.0%	0.037016	--	--	1000.0	toaa_.ACC_KERNEL@li.59
0.0%	0.032549	--	--	1000.0	toaa_.ACC_SYNC_WAIT@li.65
0.0%	0.006752	--	--	1000.0	toaa_.ACC_REGION@li.59
0.0%	0.000074	--	--	1.0	exit
0.0%	0.000031	--	--	1.0	toaa_.ACC_SYNC_WAIT@li.79
0.0%	0.000000	--	--	0.0	ETC



Performance Tools Example

Table 2: Time and Bytes Transferred for Accelerator Regions

Host Time%	Host Time	Acc Time	Acc Copy In (MBytes)	Acc Copy Out (MBytes)	Calls	Calltree
100.0%	88.749	88.697	152587.891	76293.945	5001	Total

100.0%	88.749	88.697	152587.891	76293.945	5001	toaa_

100.0%	88.749	88.697	152587.891	76293.945	5000	toaa_.ACC_REGION@li.59

3 75.5%	67.042	67.042	152587.891	--	1000	toaa_.ACC_COPY@li.59
3 24.4%	21.630	21.630	--	76293.945	1000	toaa_.ACC_COPY@li.65
3 0.0%	0.037	0.026	--	--	1000	toaa_.ACC_KERNEL@li.59
3 0.0%	0.033	--	--	--	1000	toaa_.ACC_SYNC_WAIT@li.65
3 0.0%	0.007	--	--	--	1000	toaa_.ACC_REGION@li.59(exclusive)
=====						
0.0%	0.000	--	--	--	1	toaa_.ACC_SYNC_WAIT@li.79
=====						

Processing step 3 of 3



Performance Tools Example

```
ACC: Transfer 2 items (to acc 1600000000 bytes, to host 0 bytes) from toaa2.F90:55
ACC: Execute kernel toaa_$ck_L55_1 async(auto) from toaa2.F90:55
ACC: Wait async(auto) from toaa2.F90:61
ACC: Transfer 2 items (to acc 0 bytes, to host 800000000 bytes) from toaa2.F90:61
ACC: Transfer 2 items (to acc 1600000000 bytes, to host 0 bytes) from toaa2.F90:55
ACC: Execute kernel toaa_$ck_L55_1 async(auto) from toaa2.F90:55
ACC: Wait async(auto) from toaa2.F90:61
ACC: Transfer 2 items (to acc 0 bytes, to host 800000000 bytes) from toaa2.F90:61
ACC: Transfer 2 items (to acc 1600000000 bytes, to host 0 bytes) from toaa2.F90:55
ACC: Execute kernel toaa_$ck_L55_1 async(auto) from toaa2.F90:55
ACC: Wait async(auto) from toaa2.F90:61
ACC: Transfer 2 items (to acc 0 bytes, to host 800000000 bytes) from toaa2.F90:61
ACC: Transfer 2 items (to acc 1600000000 bytes, to host 0 bytes) from toaa2.F90:55
ACC: Execute kernel toaa_$ck_L55_1 async(auto) from toaa2.F90:55
ACC: Wait async(auto) from toaa2.F90:61
ACC: Transfer 2 items (to acc 0 bytes, to host 800000000 bytes) from toaa2.F90:61
ACC: Transfer 2 items (to acc 1600000000 bytes, to host 0 bytes) from toaa2.F90:55
ACC: Execute kernel toaa_$ck_L55_1 async(auto) from toaa2.F90:55
ACC: Wait async(auto) from toaa2.F90:61
```



Performance Tools Example

```
#ifdef USE_DATA
!$acc data create(a,b)
#endif
t1 = gettimeofday()
stream_counter = 1
DO j = 1,Nchunks
  my_stream = Streams(stream_counter)
#ifdef USE_DATA
!$acc update device(a(:,j))
#endif
!$acc parallel loop
  DO i = 1,Nvec
    b(i,j) = SQRT(EXP(a(i,j)*2d0))
    b(i,j) = LOG(b(i,j)**2d0)/2d0
  ENDDO
!$acc end parallel loop
#ifdef USE_DATA
!$acc update host(b(:,j))
#endif
  stream_counter = MOD(stream_counter, 3) + 1
ENDDO
!$acc wait
t2 = gettimeofday()
!$acc end data
```

GPUでのメモリ割り当て

```
ftn -rad -hnocaf -DUSE_DATA -c -o toaa2.o toaa2.F90
ftn -rad -hnocaf -DUSE_DATA -o toaa2.x toaa2.o
pat_build -w toaa2.x
aprun toaa2.x+pat
50000944.502389029
Time = 4.1188710090027598
Experiment data file written:
/lus/scratch/beyerj/openacc/toaa/toaa2.x+pat+10178-
toaa2.x+pat
App used 1880347 resources: utime ~4s, stime ~2s
pat_rep toaa2.x+pat+10112-43t.xf
```

CPUからGPUへデータ転送

GPUからCPUへデータ転送



Performance Tools Example

Table 2: Time and Bytes Transferred for Accelerator Regions

Host Time%	Host Time	Acc Time	Acc Copy In (MBytes)	Acc Copy Out (MBytes)	Calls	Calltree
100.0%	4.148	3.714	762.939	762.939	70005	Total

100.0%	4.148	3.714	762.939	762.939	70005	toaa_
						toaa_.ACC_DATA_REGION@li.27

3	67.3%	2.792	2.487	--	762.939	30000 toaa_.ACC_UPDATE@li.71

4	60.0%	2.487	2.487	--	762.939	10000 toaa_.ACC_COPY@li.71
4	6.9%	0.286	--	--	--	10000 toaa_.ACC_SYNC_WAIT@li.71
4	0.4%	0.018	--	--	--	10000 toaa_.ACC_UPDATE@li.71(exclusive)
=====						
3	25.7%	1.066	1.055	762.939	--	20000 toaa_.ACC_UPDATE@li.52

4	25.4%	1.055	1.055	762.939	--	10000 toaa_.ACC_COPY@li.52
4	0.3%	0.011	--	--	--	10000 toaa_.ACC_UPDATE@li.52(exclusive)
=====						
[[[...]]]						

Processing step 3 of 3

Libsci_acc

- **XK向けBLAS とLAPACK**
 - 100%互換のAPI
- **CUDA、OpenACCの両方に対応**
- **FortranとCのAPIをサポート**
- **2種類のインターフェース**
 - Simpleインターフェース
 - ソースコードの変更なく、GPUの使用
 - Expertインターフェース
 - 僅かなコードの変更でGPUを有効に利用



Libsci_ACCルーチン

- BLAS

- [s,d,c,z]GEMM
- [s,d,c,z]TRSM
- [z,c]HEMM
- [s,d]SYMM
- [s,d,c,z]SYRK
- [z,d]HERK
- [s,d,c,z]SYR2K
- [s,d,c,z]TRMM
- All level 2 BLAS
- All level 1 BLAS

- LAPACK

- [d,z]GETRF
- [d,z]GETRS
- [d,z]POTRF
- [d,z]POTRS
- [d,z]GESDD
- [d,z]GEBRD
- [d,z]GEQRF
- [d,z]GELQF

Eigenvalue Solvers

- DSYEV
- ZHEEV
- DSYEVR
- ZHEEVR
- DSYEVD
- ZHEEVD
- DSYGVD
- ZHEGVD
- DGEEV
- ZGEEV

Full-HYBRID

HYBRID is planned

No HYBRID



Libsci_accのSimpleインターフェースでの使い方

メインプログラムの最初に
libsci_acc_initで初
期化

libsci_host_allocを使っ
てPinnedメモリの割り当
て

DGEMMの呼び出し方は従来
のDGEMMと一緒

行列のサイズに合わせて、
CPU, GPU, ハイブリッド実
行を選択

行列データA, B, CはCPU側

```
call libsci_acc_init()  
:  
call libsci_host_alloc(a, 8*m*lda)  
:  
  
call dgemm('n', 'n', m, n, k, alpha, &  
           a, lda, b, ldb, beta, c, ldc)
```



Libsci_accの使い方: OpenACCでGPUルーチンの実行

CPU—GPU間の データ
転送はOpenACCで
処理

DGEMMのGPU用イン
ターフェース

```
!$acc data copy(a,b,c)
```

```
!$acc parallel  
!Do Something (GPU実行)  
!$acc end parallel
```

```
!$acc host_data use_device(a,b,c)
```

```
call dgemm_acc('n','n',m,n,k,&  
               alpha,a,lda,&  
               b,ldb,beta,c,ldc)
```

```
!$acc end host_data  
!$acc end data
```



Libsci_accの使い方: OpenACCでGPUルーチンの実行

CPU-GPU間の データ
転送はOpenACCで
処理

Simpleインターフェイス
で使用

```
!$acc data copy(a,b,c)
```

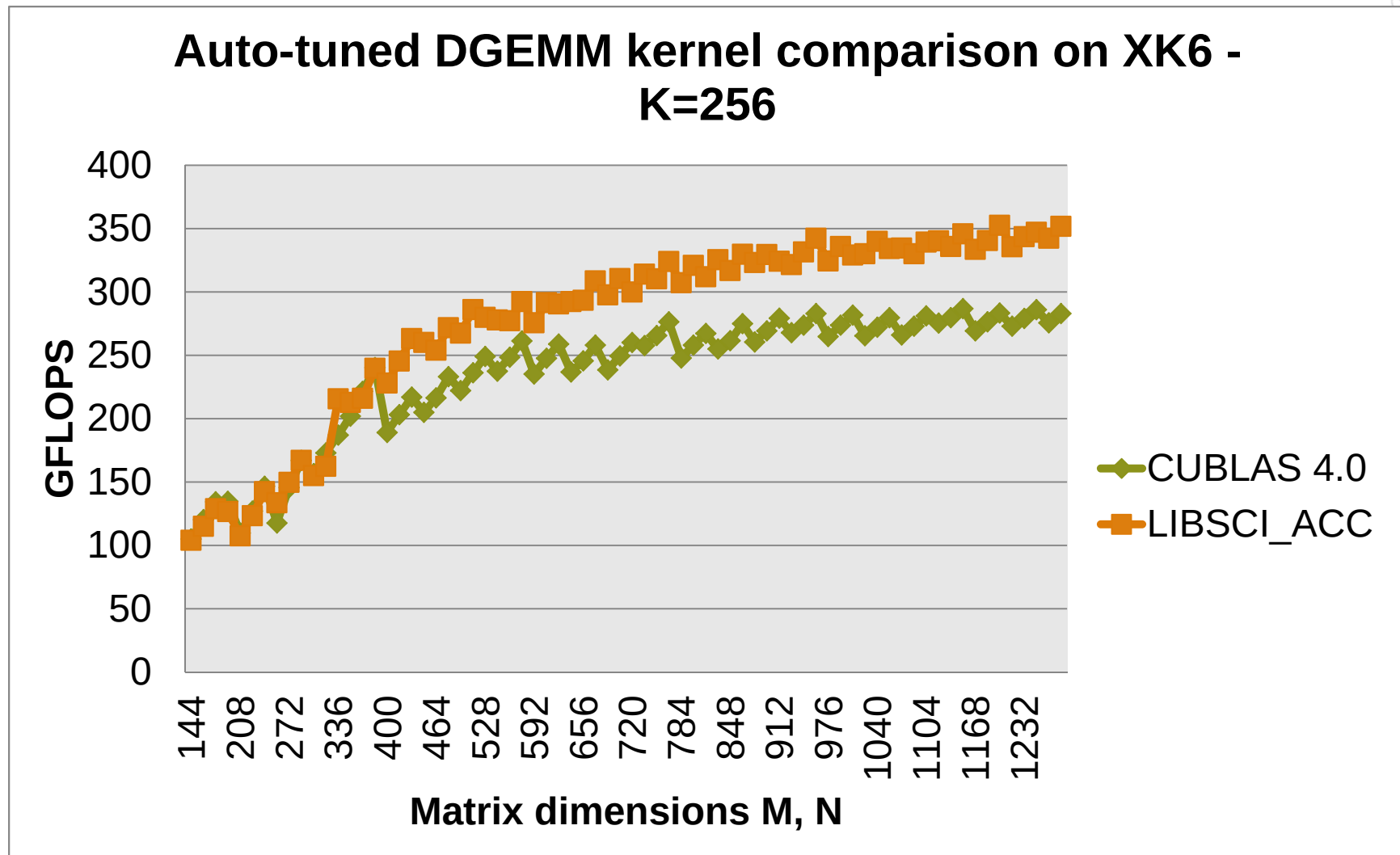
```
!$acc parallel  
!Do Something (GPU実行)  
!$acc end parallel
```

```
!$acc host_data use_device(a,b,c)
```

```
call dgemm ('n','n',m,n,k,&  
            alpha,a,lda,&  
            b,ldb,beta,c,ldc)
```

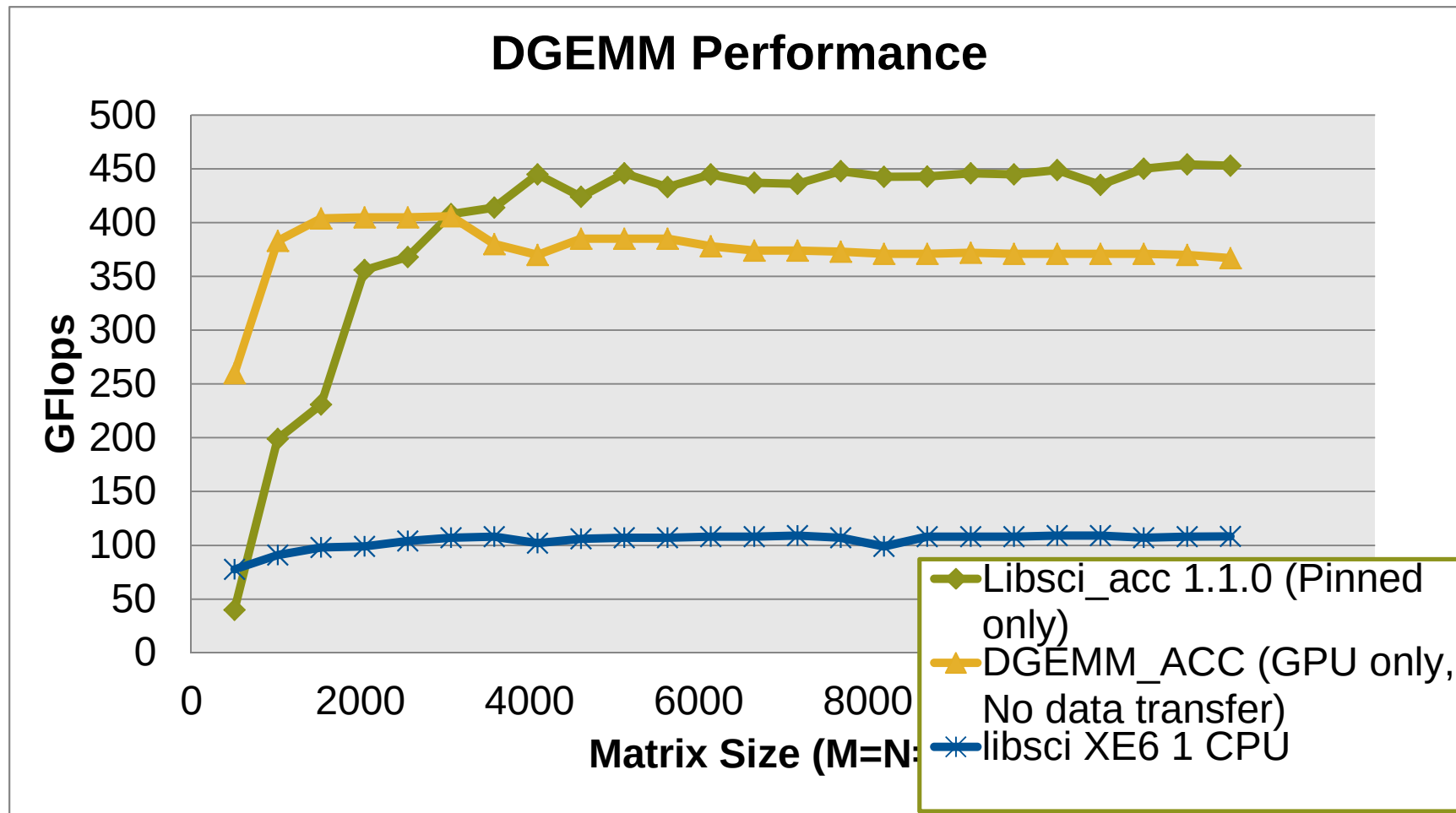
```
!$acc end host_data  
!$acc end data
```

Auto-Tuned DGEMM



CUBLAS4.1 improved performance. We are targeting to replace CUBLAS5.0 for Kepler.

DGEMMの性能



まとめ



- ヘテロジニアスマルチコア のトレンドは今後も続く
 - Fat ノードはさらにFatに
 - GPUの登場で、プログラミングはより複雑に
- アクセラレータプログラミングを効率よく行う為のツール群、研究開発
 - 高レベルなプログラミング言語のままでのアクセラレータプログラミング、性能チューニング
 - Cray Compilation Environment (CCE)
 - OpenACC のサポート
 - コンパイラによる様々な出力データをツールに読み込ませる事で更なるプログラムの最適化を可能に、
 - Cray Reveal
 - ソースコードと実行性能の対応関係の理解を容易にし、更なる性能チューニング、並列化を可能に
 - Cray Performance Analysis Toolkit
 - GPU and CPUの性能解析を1つのツールで可能に
 - Cray Auto-Tuning Libraries
 - システム、問題サイズ、入力パラメータ毎に最適化された科学計算ライブラリ