ASIP Designerを用いた RISC-Vステレオマッチングプロセッサのご紹介

RISC-V Days Tokyo 2022 Spring

日本シノプシス合同会社
スタッフアプリケーションエンジニア 伴野 充

2022年5月31日

アジェンダ

- ASIP Designerのご紹介
- RISC-Vプロセッサの開発
- Tmatch: RISC-Vステレオマッチングプロセッサのご紹介
- まとめ

ASIP Designerのご紹介

ドメインに特化したプロセッサの時代

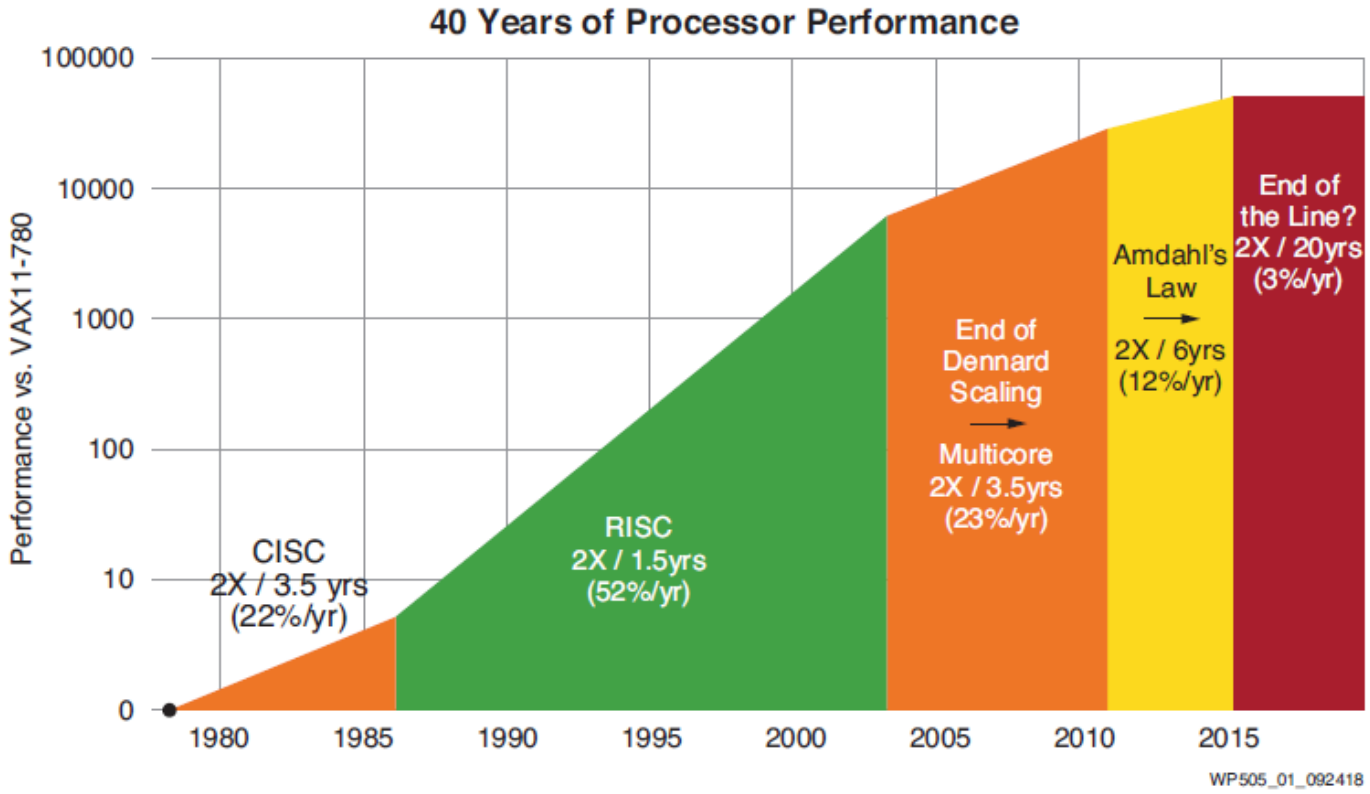


Figure 1: **Processor Performance vs. Time**

Source: Xilinx Whitepaper 505 – Versal ACAP
(reference to J. Hennessy, D. Patterson, *Computer Architecture: A Quantitative Approach* (6th Edition, 2019)

- 1995 – 2003:

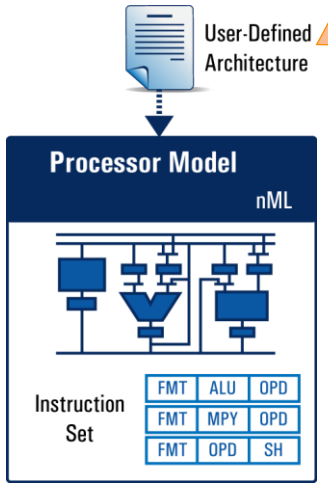
  プロセス技術による性能（より高い周波数だが同じ電力消費）およびメモリアクセスの最適化（Dennard Scaling）
- 2003: Multicore

  タスクの並列化制限に到達（Amdahlの法則）
- 2015: Domain-Specific Processors

  ヘテロジニアス・マルチコア設計への移行

ASIP Designer

- アーキテクチャ構造記述言語(nML)により、
  - **アプリケーションに特化した命令セットを持つ専用プロセッサ/ DSP を自動生成する設計環境**



nML記述

```
action {
  stage E1:
    vecd =
     vsub(vecr=V[r],vecs=V[t])@vec;
    V[t] = vect = vabs(vecd) @vabs;
  }
  syntax : "vadiff v"t ",v"r ",v"t;
  image  : t::r;
```

ASIP設計開発ツール
“ASIP Designer”

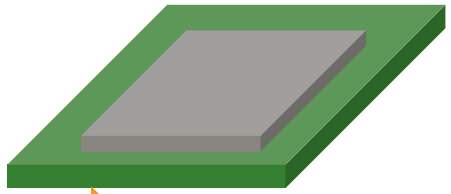
命令セット
シミュレータ(ISS)

SystemC ベース
System Simulator

C/C++ コンパイラ
アセンブラ/リンカー

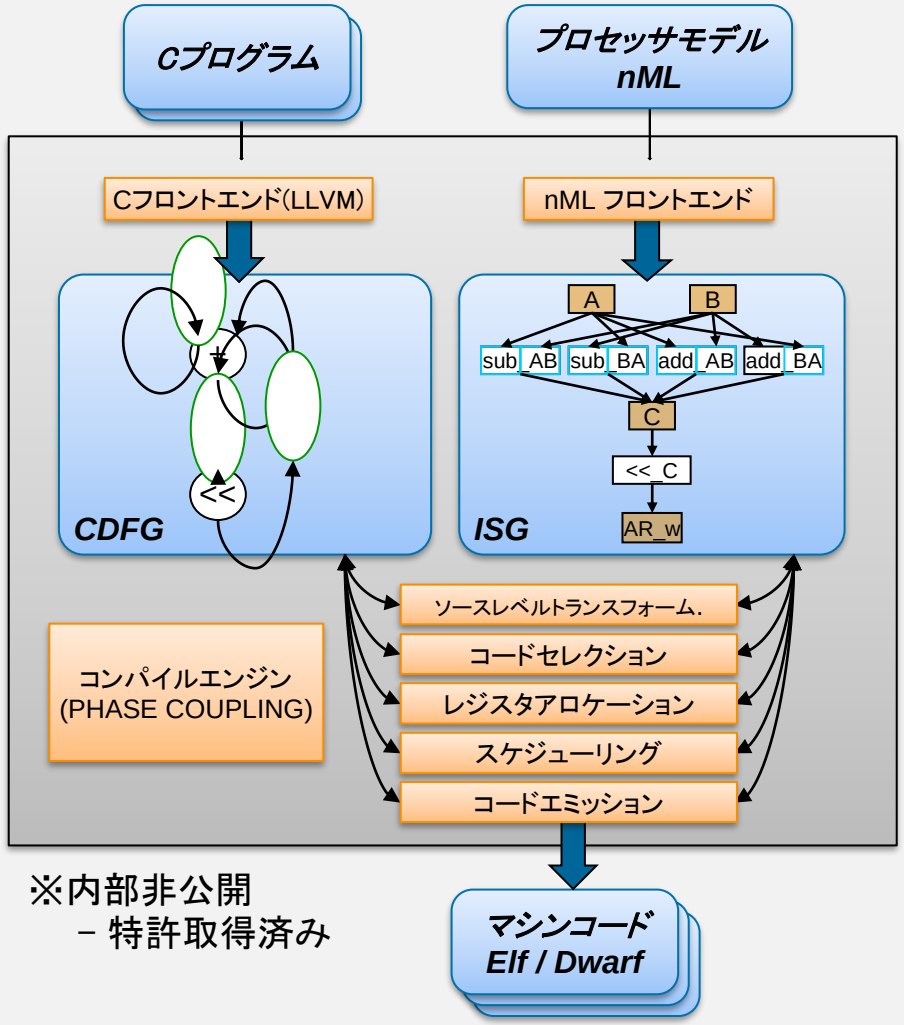
デバッガ/プロファイラ

Verilog / VHDL
RTL コード



ASIC / FPGA

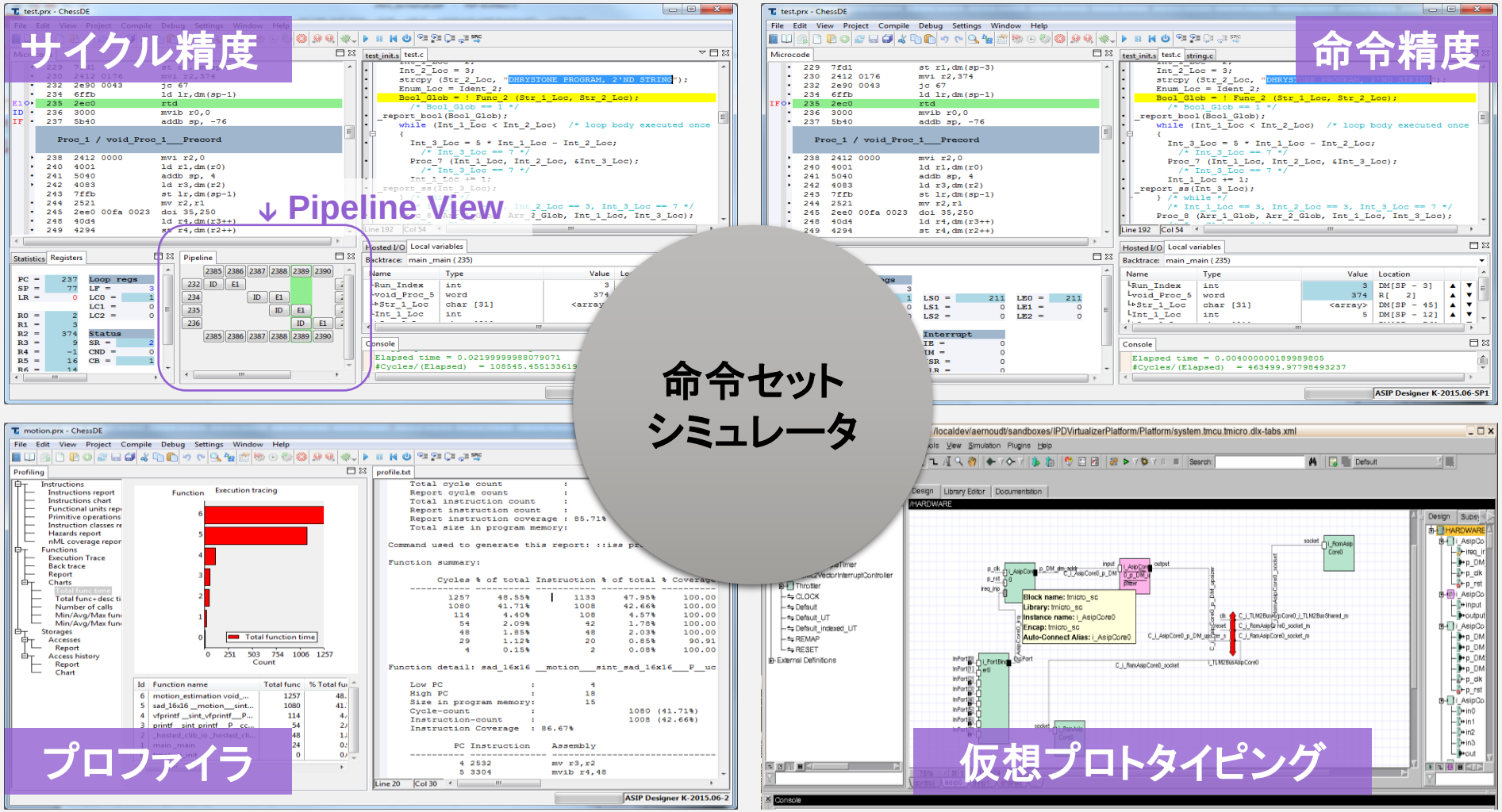
Graph-Based C コンパイラ “Chess”



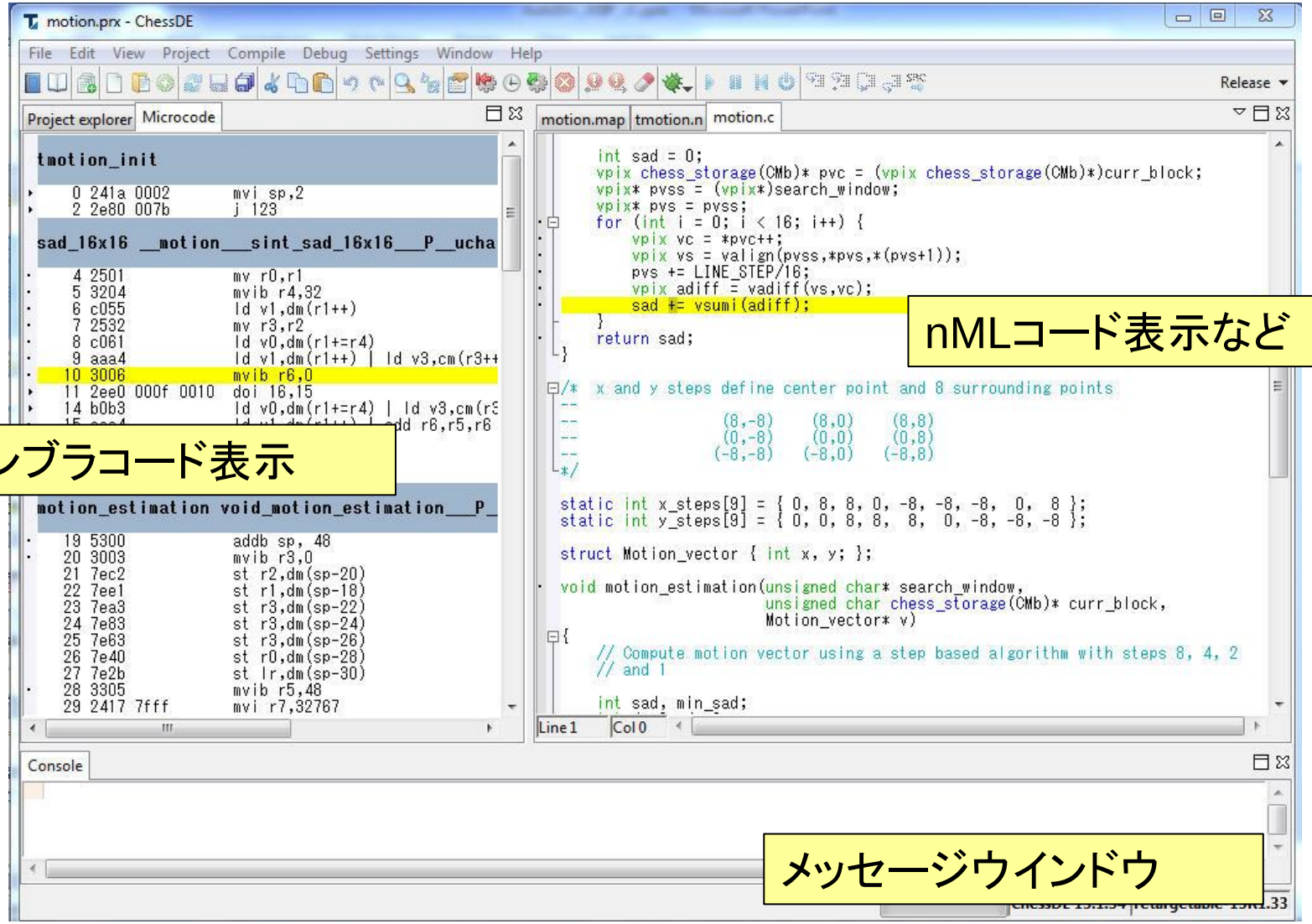
※内部非公開
– 特許取得済み

- フロントエンド
  - C → コントロールデータフローグラフ
  - nML → インストラクションセットグラフ
- コンパイルフェーズ
  - CDFGから ISGへマッピング
  - 独自“Graph”アルゴリズム
- ISGに含まれる構成情報
  - ハードウェアリソース
  - データタイプ
  - 接続情報
  - 命令エンコード情報
  - 命令レベルの並列化情報
  - 命令パイプライン情報

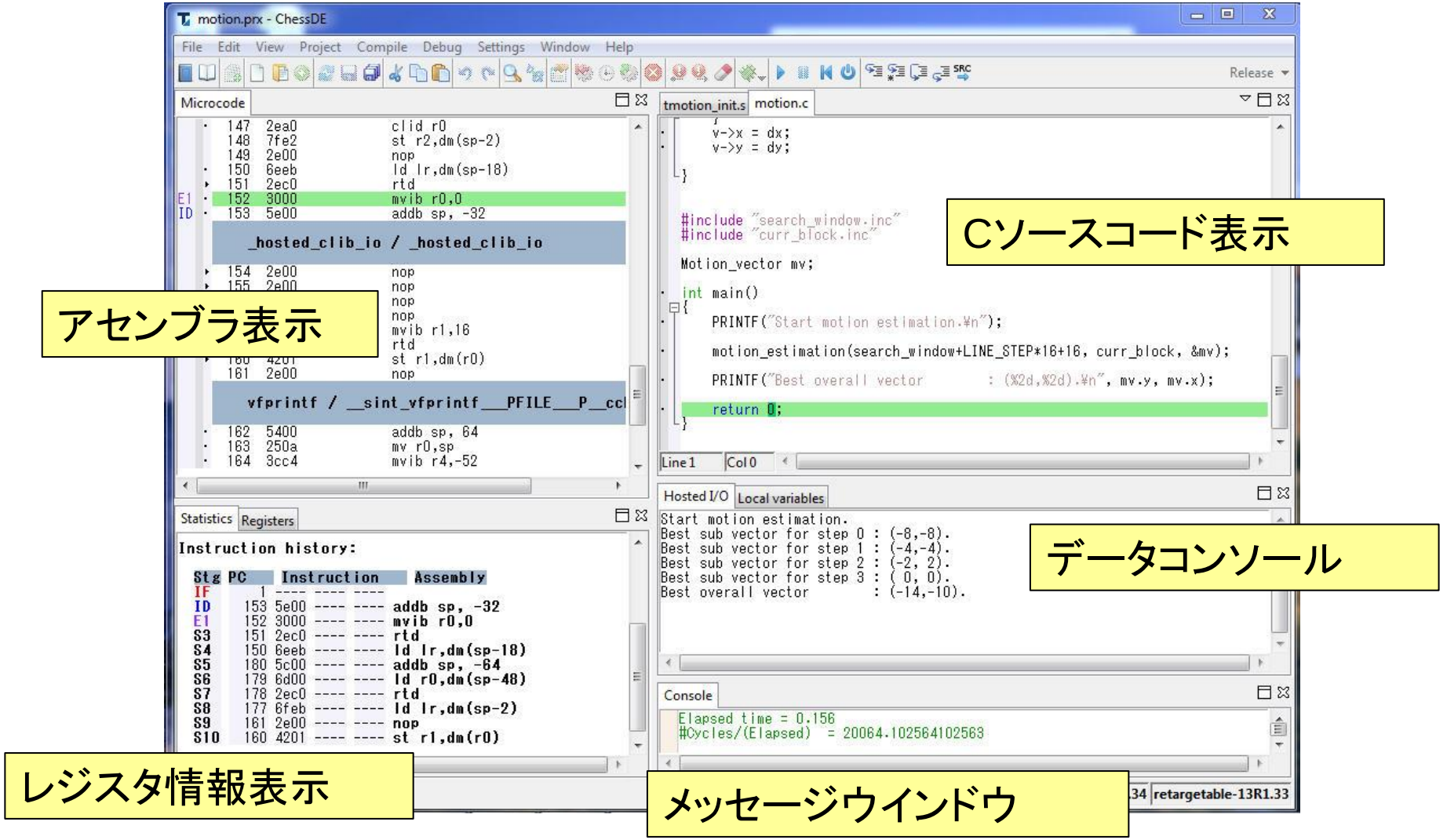
ASIP Designer - 命令セットシミュレータ



ASIP Designer生成 ー ASIPデバッガ



ASIP Designer生成 ー ソフトウェアデバッガ



ソフトウェア・プロファイリング機能

命令プロファイリング

```
Function summary:

      Cycles % of total Instruction % of total % Coverage
------------ ---------- ----------- ---------- ----------
        1620     51.01%        1548     52.60%     86.67%
        1258     39.61%        1133     38.50%     90.38%
         131      4.12%         124      4.21%     94.74%
          63      1.98%          49      1.66%     77.78%
          56      1.76%          56      1.90%    100.00%
          38      1.20%          27      0.92%     70.97%
           7      0.22%           4      0.14%     75.00%

Function detail: sad_16x16__motion__sint_sad_16x16__P_u

    Low PC                :          4
    High PC               :         18
    Size in program memory:         15
    Cycle-count           :                  1620 (51.01%)
    Instruction-count     :                  1548 (52.60%)
    Instruction Coverage  : 86.67%

           PC Instruction    Assembly
---------- ---------- ----------------------------------------
            4 2501           mv r0,r1
            5 3204           mvib r4,32
            6 c055           ld v1,dm(r1++)
            7 2532           mv r3,r2
            8 c061           ld v0,dm(r1+=r4)
            9 aaa4           ld v1,dm(r1++) | ld v3,cm(r3++)
           10 3006           mvib r6,0
           11 2ee0 000f 0010 doi 16,15
           14 b0b3           ld v0,dm(r1+=r4) | ld v3,cm(r3
           15 aac4           ld v1,dm(r1++) | add r6,r5,r6
           16 2ec0           rtd
           17 2506           mv r0,r6
           18 2e00           nop

Function detail: motion_estimation void_motion_estimation___P_

    Low PC                :         19
    High PC               :        122
    Size in program memory:        104
    Cycle-count           :                  1258 (39.61%)
    Instruction-count     :                  1133 (38.50%)
    Instruction Coverage  : 90.38%
```

```
                                                    Relative cycle use in simulation
------------------------------------------------ ----------------------------------------
_sint_sad_16x16___P__uchar___P **************************
_d_motion_estimation___P__uchar **********************
ntf___PFILE___P__cchar___Pvoi **
__P__cchar
ed_clib_io

Exe-count      Cycles Wait states Relative cycle use within func
---------- ---------- ---------- ----------------------------------------
        36         36          0 ***
        36         36          0 ***
        36         36          0 ***
        36         36          0 ***
        36         36          0 ***
        36         36          0 ***
        36         36          0 ***
        36        108          0 *********
       576        576          0 ********************************
       576        576          0 ********************************
        36         36          0 ***
        36         36          0 ***
        36         36          0 ***

_PMotion_vector
```

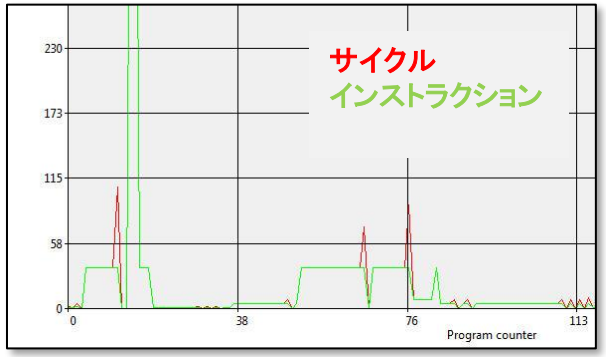
命令＆サイクル・カウント

```
ISS command      = <PROCDIR>/../iss/tmotion_ca
ISS mode         = Cycle accurate

Cycle count        = 3176
Instruction count  = 2943

PC =   130

SP =    98
Stack area: DMb[    2 ..  2049], growing up
Maximum stack pointer value =   146

Instruction history:

 Stg PC   Instruction    Assembly
 IF   130 ---- ---- ----
 ID   180 5c00 ---- ---- addb sp, -64
 E1   179 6d00 ---- ---- ld r0,dm(sp-48)
 S3   178 2ec0 ---- ---- rtd
 S4   177 6feb ---- ---- ld lr,dm(sp-2)
 S5   161 2e00 ---- ---- nop
 S6   160 4201 ---- ---- st r1,dm(r0)
 S7   159 2ec0 ---- ---- rtd
 S8   158 3101 ---- ---- mvib r1,16
 S9   157 2e00 ---- ---- nop
 S10  156 2e00 ---- ---- nop
```

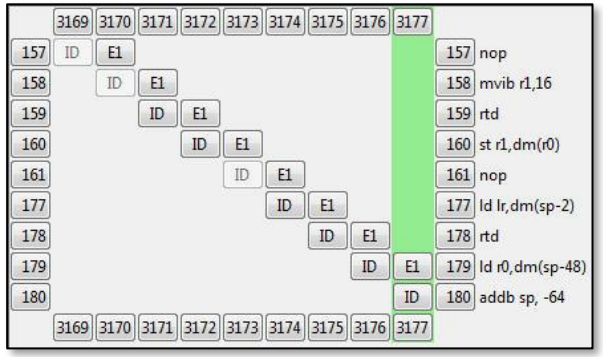
nMLカバレッジ

```
tmotion  (0.266224)
 .alu_instr  (0.0654762)
  .alu_rrr  (0.142857)
   +alu_op  (0.142857)
     -add  (1)
     -addc  (0)
     -sub  (0)
     -subb  (0)
     -and  (0)
     -or  (0)
     -xor  (0)
  .compare_rr  (0.25)
   +compare_op  (0.25)
     -lt  (1)
     -ltu  (0)
     -le  (0)
     -leu  (0)
     -gt  (0)
     -gtu  (0)
     -ge  (1)
     -geu  (0)
  .equal_rr  (0)
  .alu_rr  (0)
  .minmax_rrr  (0)
  .select_rrr  (0)
 .shift_instr  (0.333333)
  .shift_rrr  (0.333333)
```

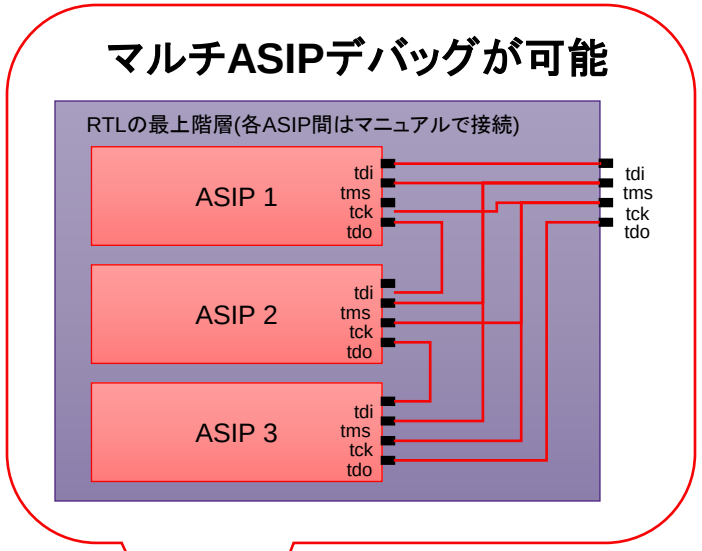
命令＆サイクル・チャート



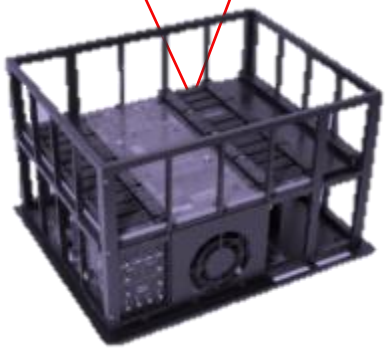
パイプラインビュー



RTLコード生成と実機デバッグ環境の構築



- nML(PDG)記述からRTLコードを生成
  - Verilog/VHDL RTLコード
  - JTAG Key/JTAG key2をサポート
  - Design Compiler/ Simplify用スクリプト
  - **生成プロセッサ（RTL）のロイヤリティ不要**



HAPS-80 FPGAボード
シリーズ

Xilinx USB2

Digilent JTAG HS2

Macraigor

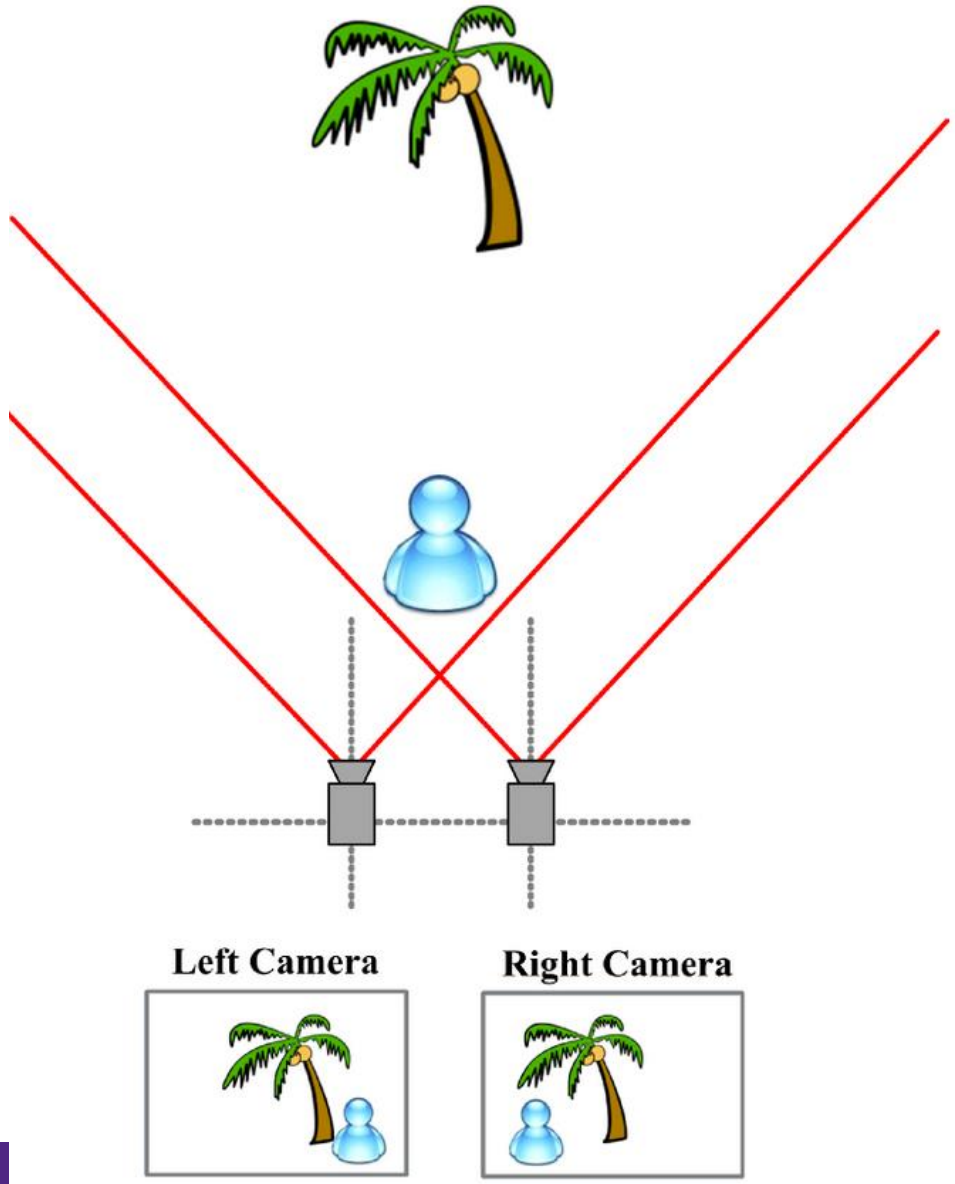
Lauterbach

FTDI USB Hi-Speed to MPSSE cable.
Segger J-Link Ultra cable.

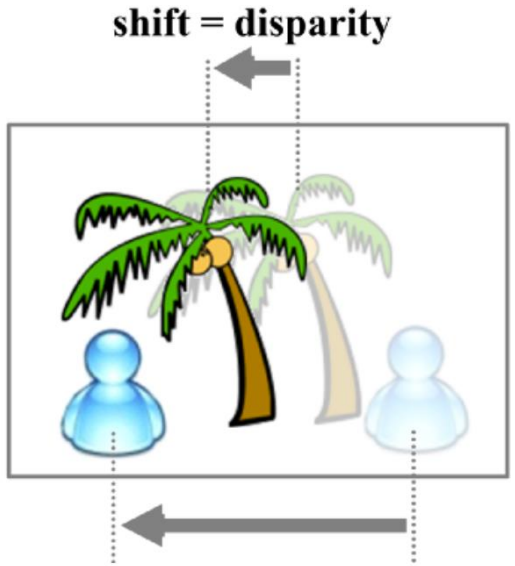
ソフトウェア
デバッガ

Tmatch: RISC-Vステレオマッチングプロセッサのご紹介

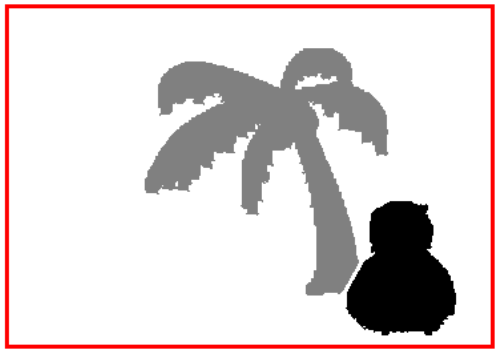
ステレオイメージの視差マップ



- 視差は水平方向のシフト

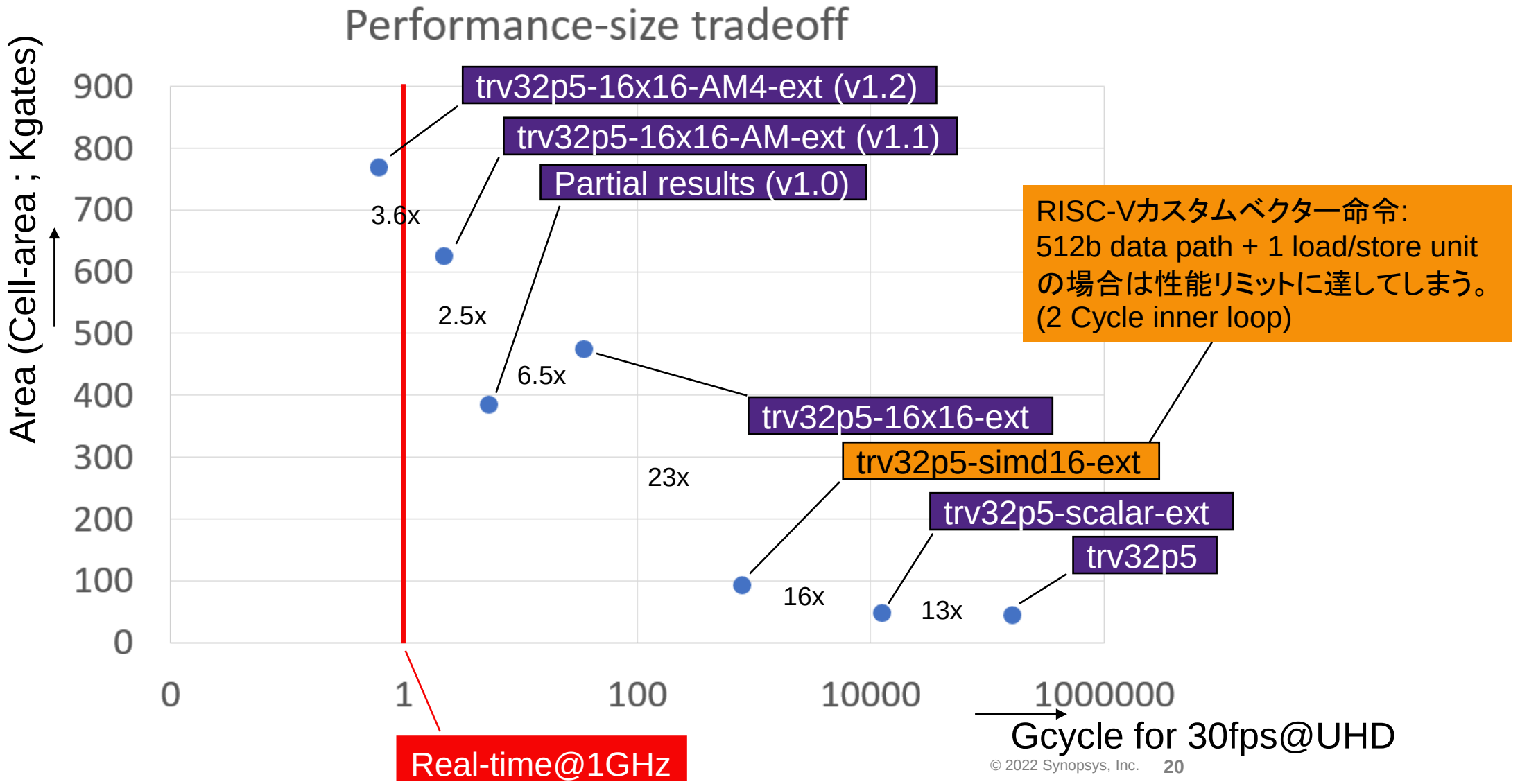


- 各ピクセルに対して計算された視差マップ

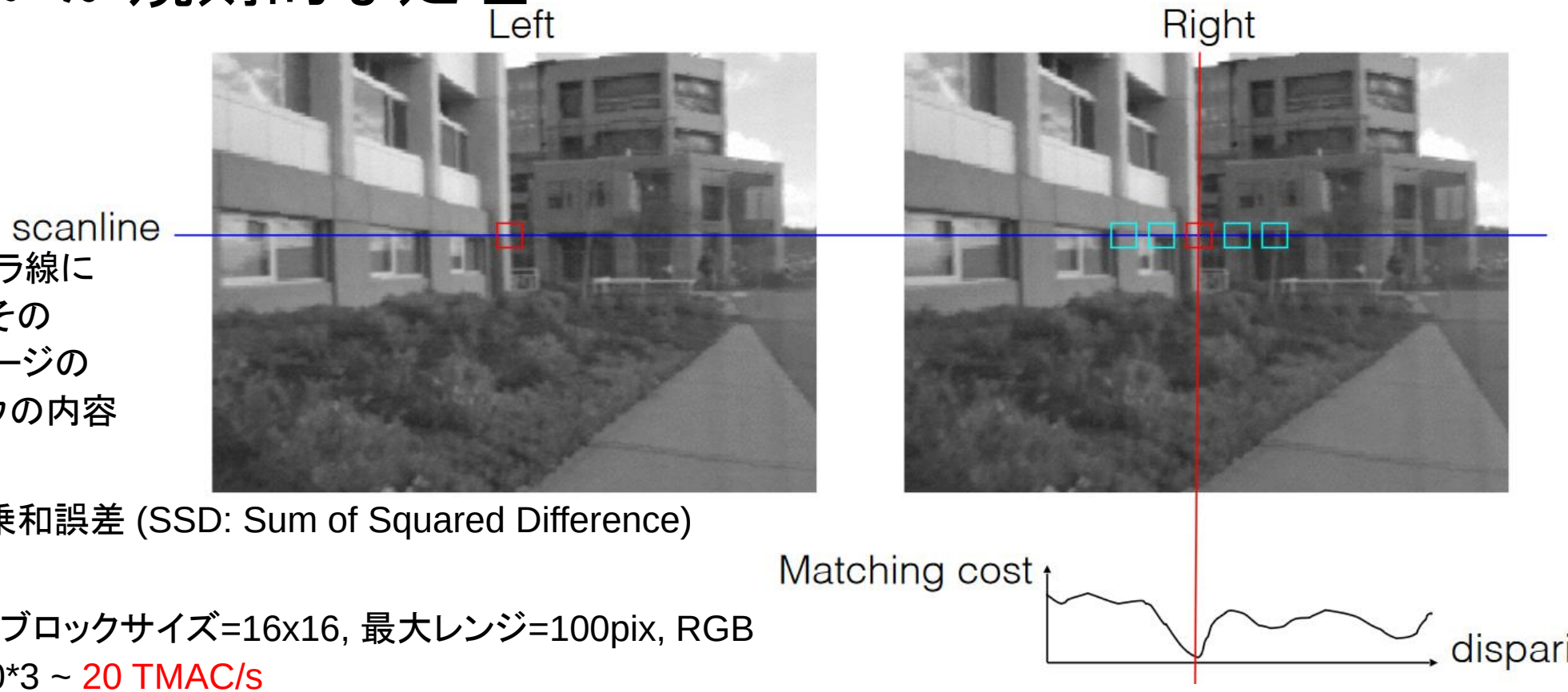


- 奥行の計算に使用

アーキテクチャ探索の軌跡



演算量は多いが規則的な処理



- ウィンドウをエピポーラ線にそってスライドさせ、そのウィンドウと左側イメージのリファレンスウィンドウの内容を比較する
- マッチングコスト: 二乗和誤差 (SSD: Sum of Squared Difference)
- 4Kイメージ, >30fps, ブロックサイズ=16x16, 最大レンジ=100pix, RGB
  →8M*30*16*16*100*3 ~ 20 TMAC/s
  →20K mults/cycle @1GHz
- アーキテクチャ最適化: 専用命令, 命令レベル並列性, ベクタライゼーション, マルチコア, メモリ? → デザインスペースの探索
- アプリケーション最適化: 再利用, アルゴリズム? → アーキテクチャ/アプリケーションの協調最適化
- 規則的なアルゴリズム

trv32p5xにおけるコンパイル結果

理想的なマルチコア並列化でも166000コアが必要!

```
7388   4   c.nop
7390   5   mv      x30, x8
7394   5   li      x13, 128
7398   5   li      x8,  128
7402   5   li      x9,  128
7406   5   bltz    x26, 60
7410   5   bge     x26, x19, 56
7414   5   lw      x13, 0(x5)
7418   5   mv      x12, x26
7422   5   bgtz    x26, 6
7426   5   c.li    x12, 0
7428   5   blt     x12, x19, 6
7432   5   c.lwsp  x12, 68(x2)
7434   5   c.add   x12, x25
7436   5   mul     x12, x4,  x12
7440   5   addi    x9,  x12, 1
7444   5   add     x8,  x9,  x13
7448   5   c.addi  x9,  1
7450   5   c.add   x12, x13
7452   5   c.add   x13, x9
7454   5   lbu     x9,  0(x12)
7458   5   lbu     x8,  0(x8)
7462   5   lbu     x13, 0(x13)
7466   5   mv      x15, x29
7470   5   lbu     x12, -1(x15!)
7474   5   lbu     x14, 0(x15)
7478   5   c.and   x9,  x11
7480   5   c.sub   x14, x9
7482   5   c.and   x8,  x11
7484   5   c.sub   x12, x8
7486   5   lbu     x9,  2(x15)
7490   5   c.and   x13, x11
7492   5   mul     x8,  x14, x14
7496   5   c.sub   x9,  x13
7498   5   c.add   x8,  x30
7500   5   mul     x13, x12, x12
7504   5   mul     x30, x9,  x9
7508   5   c.add   x8,  x13
7510   5   c.add   x29, x4
7512   5   c.addi  x26, 1
7514   5   c.add   x8,  x30
7516   4   c.lwsp  x26, 128(x2)
```

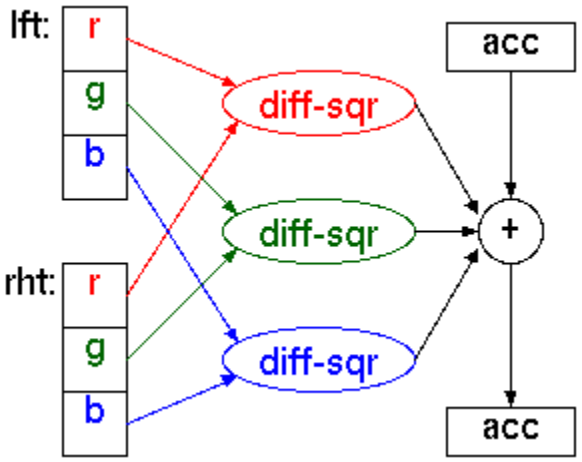
```
44  for (int i=0+half_block_size; i<left->h-half_block_size; i++)
45  {
46    for (int j=0+half_block_size; j<left->w-half_block_size; j++)
47    {
48      if (j+disparity_min-half_block_size<0 || j+disparity_max+half_block_size>COL -1) {
49        for (int range=disparity_min; range<=disparity_max; range++)
50        {
51          SSD = 0;
52
53          for (l_r = -half_block_size + i; l_r <= half_block_size + i; l_r++)
54          {
55            for (l_c = -half_block_size + j; l_c <= half_block_size + j; l_c++)
56            {
57              r_r = l_r;
58              r_c = l_c + range;
59
60              if ((r_c < 0) || (r_c > (COL -1))) {
61                pix[0] = 128;
62                pix[1] = 128;
63                pix[2] = 128;
64              }
65              else {
66                pix[0] = right->data[r_r*COL*CH+std::min(std::max(0, r_c), COL-1)*CH];
67                pix[1] = right->data[r_r*COL*CH+std::min(std::max(0, r_c), COL-1)*CH+1];
68                pix[2] = right->data[r_r*COL*CH+std::min(std::max(0, r_c), COL-1)*CH+2];
69              }
70
71              SSD += square(left->data[l_r*COL*CH+l_c*CH] - pix[0])
72                   + square(left->data[l_r*COL*CH+l_c*CH+1] - pix[1])
73                   + square(left->data[l_r*COL*CH+l_c*CH+2] - pix[2]);
74            }
75
76          }
77
78          if (SSD < SSD_value[i*COL+j] - THRESHOLD)
79          {
80            disp[i*COL+j] = range;
81            SSD_value[i*COL+j] = SSD;
82          }
83        }
84      }
      else {
```

26 cycles (avg)

- 166 Tcyc/s
- @1GHz → 166 Kcores
- 非現実的

ストレートフォワードな専用命令とRGBベクタライゼーション

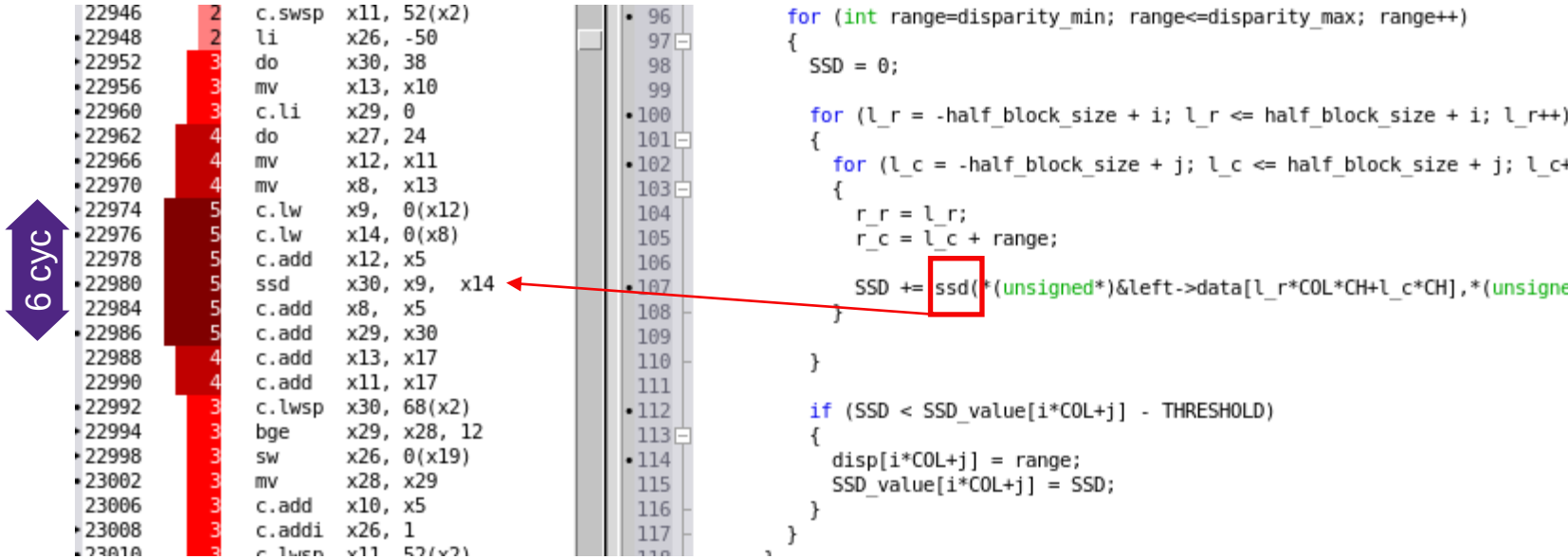
13倍の高速化, 依然として非効率的

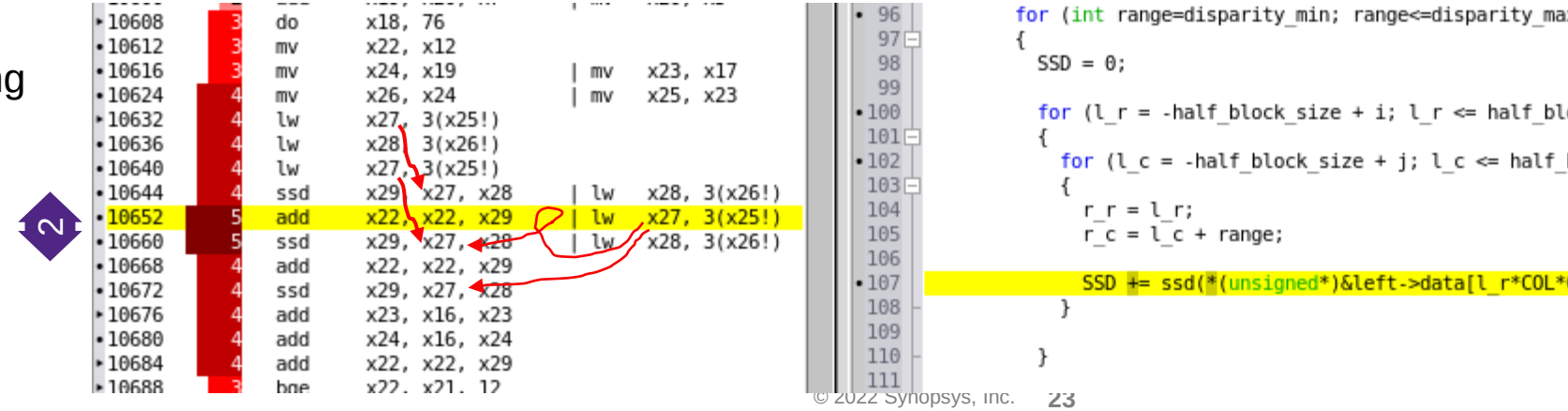


- SIMD3: RGB pixel in w32
- diff-sqr-acc (専用命令)
- 2 cycle innerloopが可能:
  - Zloop + aggressive software pipelining
  - ILP: Load || compute

→ 6.4 Titer/s → 12.8 Tcyc/s

- @1GHz → 12.8 Kcores
- 依然として非現実的

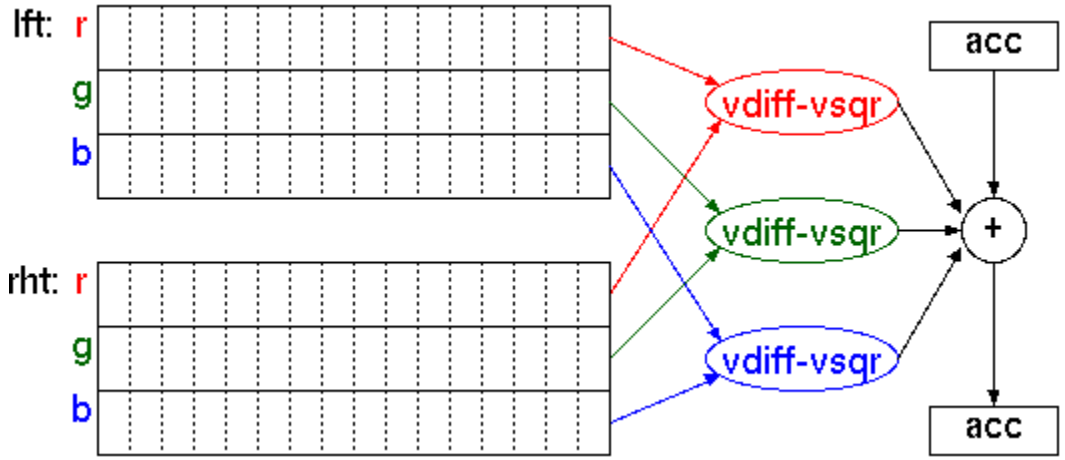




さらなるベクタライゼーション

16倍高速化, しかしまだ非効率

- SIMD3x16 → 24x16=384b vectors
- 特化したベクター diff-sqr-acc
- 2 cycle innerloopが可能:
  - Zloop + software piplining
  - ILP: Load || compute
- 16倍高速: 400 Giter/s → 800 Gcyc/s
- @1GHz → 800 cores
- 依然として非現実的
- 次のステップは?
  - メモリアクセス効率化のための専用シフトレジスタの追加
  - シングルサイクルの16x16 SSD演算命令 (23倍高速化)
  - データを再利用できる専用メモリの追加
  - 5-way VLIW

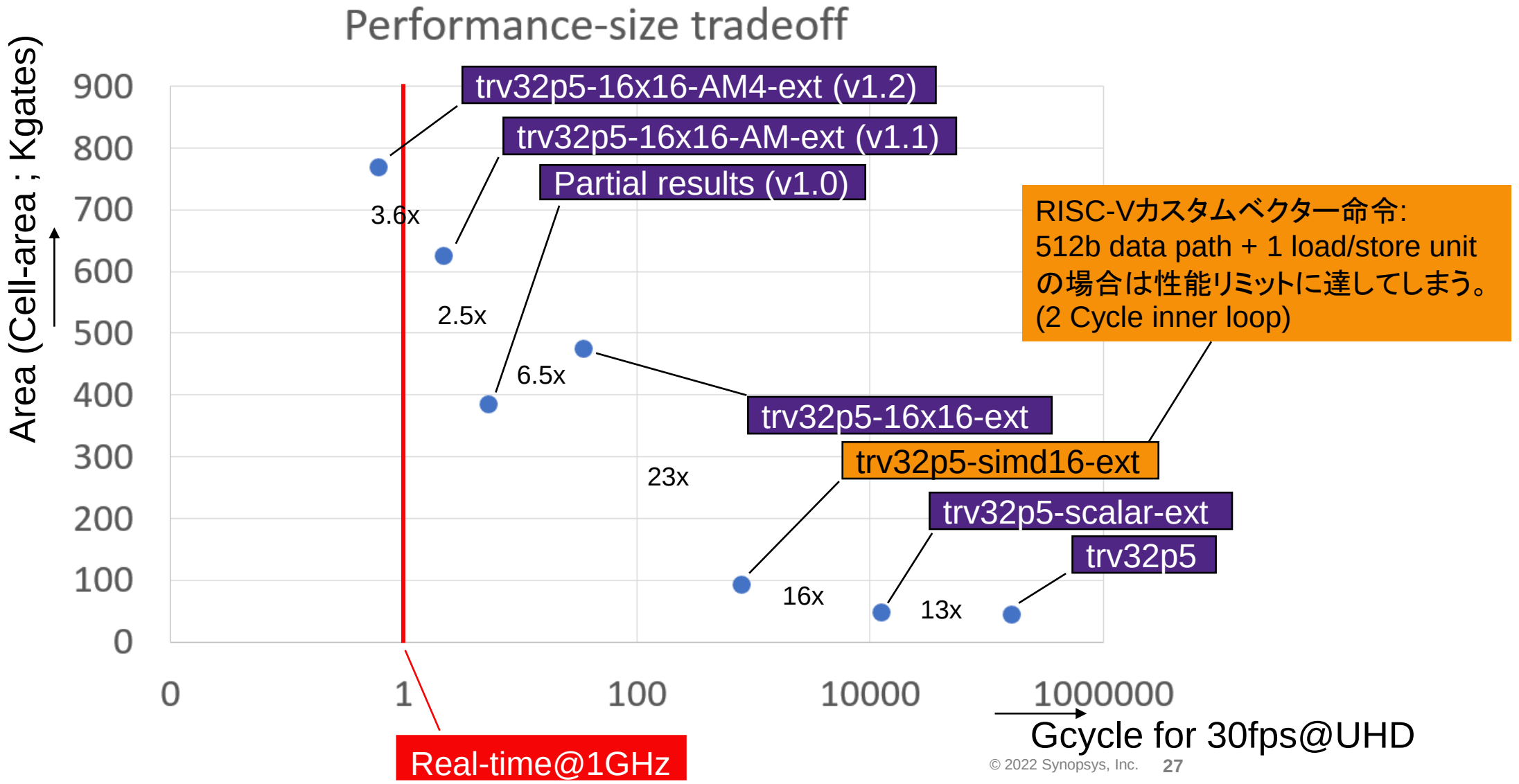



```
14952   2   nop                  | lw   x27, 0(x7)
14960   3   addi   x29, x5,  0   | nop
14968   3   addi   x30, x25, 0   | nop
14976   3   addi   x28, x16, 0   | nop
14984   3   do     x23, 32       | nop
14992   3   nop                  | lv   v1,  x14(x30!)
15000   3   nop                  | lv   v0,  x9(x29!)
15008   4   nop                  | lv   v0,  x9(x29!)
15016   4   ssd    x28+=v0,  v1  | lv   v1,  x14(x30!)
15024   3   addi   x0,  x0,  0   | nop
15032   3   bge    x28, x27, 24  | nop
15040   3   nop                  | sw   x26, 0(x6)
15048   3   addi   x27, x28, 0   | nop
15056   3   addi   x26, x26, 1   | nop
15064   3   addi   x25, x25, 1   | nop
15072   2   nop                  | sw   x27, 4(x7!)
15080   2   addi   x24, x24, 1   | nop
```

2

```
128         pix_t chess_storage(VM)* p_lpixel = &lpixels[(i-half_block_s
129         pix_t chess_storage(VM)* p_rpixel = &rpixels[(i-half_block_s
130
131         int row_inc = 2;
132
133         for (l_r = -half_block_size + i; l_r < half_block_size + i;
134         {
135           //Note: block size must be integer multiple of Vector leng
136           for (int s = 0; s < 2*half_block_size; s+=VSIZE) {
137             SSD = pix_ssdac(SSD,*(vpix_t chess_storage(VM)*)(p_lpixe
138           }
139
140           p_lpixel+=COL*row_inc; p_rpixel+=COL_exp*row_inc;
141
142         }
143
144
```

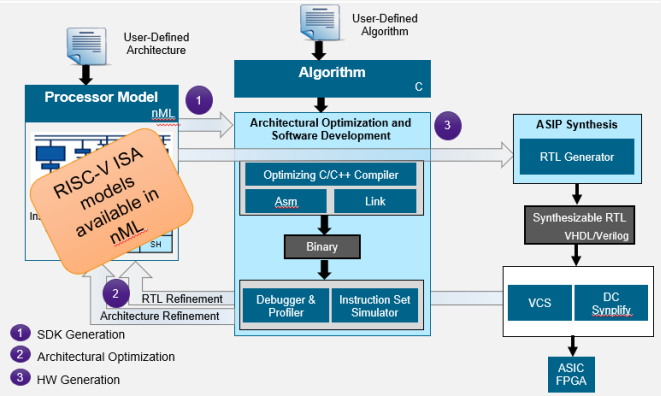
アーキテクチャ探索の軌跡

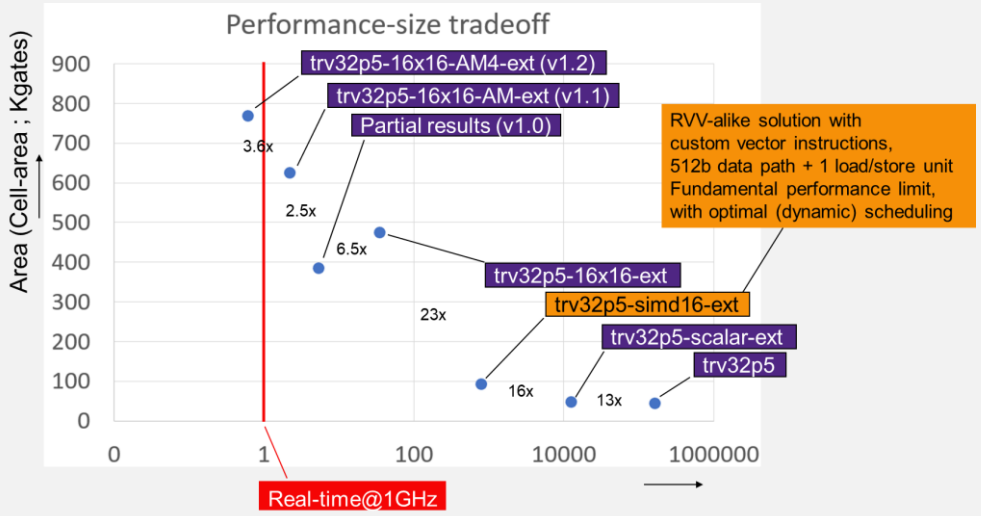


まとめ

ASIP Designer – ASIP デザインを容易に

- アプリケーションに特化したアーキテクチャによるPPAの大幅な改善
- RISC-V ISAベースのExampleモデルで即座にトライアルが可能
- 新しいExampleモデルであるTmatchステレオマッチングプロセッサをご紹介
- **製品の差別化を実現する拡張可能かつロイヤリティフリーなRISC-Vプロセッサを開発！**
  - ご興味のある方はぜひご連絡ください
  - コンタクト先: jp-mkg_info@synopsys.com





SYNOPSYS®
Silicon to Software™

Thank You