NSI-TEXE

NSITEXE次世代RISC-Vプロセッサへの取組み

(株)NSITEXE　西村 成司

アジェンダ

1. **会社紹介**
2. RISC-Vと可変長ベクトル命令
3. NSITEXEの次世代RISC-Vプロセッサ向けSDKの構想
4. 何故、SYCLが注目されるのか
5. まとめ

NSI-TEXE

会社紹介

- **半導体IP、半導体ソリューションカンパニー**
  - 2017年 (株)デンソーの出資により設立
  - 世界各地から集まった60名以上のエンジニアが在籍
- **エンベデッドシステム向け高効率半導体IP（DFP）を開発**
  - 2020年1月 最初のDFP製品DR1000Cをリリース
  - RISC-Vプロセッサ、AIエンジン、セキュリティサブシステム等も開発
- **エコシステム**
  - 自動運転等の自動車向けシステム、ロボティクス、FA、IoT等、次世代を担う様々な分野のアプリケーションをサポート
  - HW/SW、デザインメソドロジ、ソリューション各社との幅広いパートナーシップ

https://www.nsitexe.com/

アジェンダ

1. 会社紹介
2. **RISC-Vと可変長ベクトル命令**
3. NSITEXEの次世代RISC-Vプロセッサ向けSDKの構想
4. 何故、SYCLが注目されるのか
5. まとめ

NSI-TEXE

今、RISC-Vが注目されています

NSI-TEXE

- オープンでロイヤリティフリーな命令セットアーキテクチャ
  - 過去のしがらみに捕らわれない、新規設計
  - NSITEXE、SiFiveなど実際に動作するプロセッサが既に存在
- SIMDの先を行く、可変長ベクトル命令
  - かつてのスーパーコンピュータで主流だった技術
  - メモリ遅延の隠蔽が得意で高い実効効率を発揮

ソフトウエアから見る固定長SIMDと可変長ベクトルの違い

NSI-TEXE

```
int i;

for (i = 0; i < N; i++) {
    y[i] = a * x[i] + y[i];
}
```

長さ4のSIMD命令

```
int i;
const int vl = 4;

for (i = 0; i <= N - vl; i += vl) {
    float4 vx = vload4(0, x + i),
           vy = vload4(0, y + i);
    vy = a * vx + vy;
    vstore4(vy, 0, y + i);
}

for ( ; i < N; i++) { // 余りの処理部分
    y[i] = a * x[i] + y[i];
}
```

float4 長さ4のfloatデータを指すSIMDデータ型
vload4 長さ4のSIMDデータをメモリから読出す組込み関数
vstore4 長さ4のSIMDデータをメモリに書込む組込み関数

可変長ベクトル命令

```
int i;
int vl = set_vl(MAX_VL);

for (i = 0; i < N; i += vl) {
    if (vl > N - i)    // ベクトル長の調整
        vl = set_vl(N - i);
    vfloat vx = vload(0, x + i),
           vy = vload(0, y + i);
    vy = a * vx + vy;
    vstore(vy, 0, y + i);
}
```

MAX_VL アーキテクチャ上の最大ベクトル長を指す定数
set_vl ベクトル長を設定する組込み関数
vfloat floatデータを指すベクトル型
vload ベクトルデータをメモリから読出す組込み関数
vstore ベクトルデータをメモリに書込む組込み関数

可変長ベクトル命令を用いる事のメリット

NSI-TEXE

	固定長SIMD命令	可変長ベクトル命令
主目的	演算器の理論ピーク性能の向上	演算器の稼働率の向上
並列度	≦16	≦256
命令セット	複雑 SIMD長毎に命令が存在	シンプル 命令はベクトル長に依存しない
メモリ遅延の対応	キャッシュメモリ（キャッシュライン）とプリフェッチ	ベクトルロード・ストアで長いベクトルデータを一斉にアクセス
ランダムメモリアクセス	弱い 連続メモリアクセスに最適化されたHW	強い ギャザースキャッタでHW的に対応
演算器の実行効率改善	スーパースカラとOoO実行 マルチスレッド	ベクトル命令のチェイニング デカップルドアーキテクチャ マルチスレッド
条件分岐対応	真・偽両方の値を計算してSELECT命令でマージ	マスク命令がサポートされる

RISC-Vの抱えるソフトウエアの課題

NSI-TEXE

- 「過去のしがらみに捕らわれない」＝「実績がない」
  - 商用レベルでのサポートと品質保証
- 可変長ベクトル命令に対応する並列化ツール
  - マルチコア対応に加えて、更なる並列性を引き出す事
- 既存アプリケーション資産の移植性
  - 顧客のアプリケーション資産を活用できる事

アジェンダ

1. 会社紹介
2. RISC-Vと可変長ベクトル命令
3. **NSITEXEの次世代RISC-Vプロセッサ向けSDKの構想**
4. 何故、SYCLが注目されるのか
5. まとめ

NSI-TEXE

NSITEXEの次世代RISC-Vプロセッサ向けSDKの構想

NSI-TEXE

OpenMPディレクティブが挿入された標準Cコード

```
#pragma omp target map(to:A) map(from:B)
while (error > tol && ++iter < iter_max) {
    error = 0.0;
#pragma omp parallel for collapse(2) reduction(max:error)
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            B[i][j] = 0.25 * (A[i-1][j] + A[i][j-1] +
                              A[i][j+1] + A[i+1][j]);
            error = max(error, fabs(B[i][j] - A[i][j]);
        }
    }
...
}
```

OpenMP

SYCL™

ホストコードとカーネルコードが同一ソースコード上で記述されるC++環境

OpenCL™

アクセラレータ基盤

RVV命令を備える次世代DFP

RISC-V

KHRONOS GROUP

OpenCLを核に据える、**ベンダニュートラルで標準的なプログラミング環境**を実現

ユーザのアプリケーション資産への投資を活かすためにも**標準準拠**が重要

高位インターフェースの想定されるユースケース

- 検証済みのコード資産の活用

マルチコア未対応

既存のコード資産

ユーザからの強い要求

コード資産には手を付けたくない！！

OpenMP

ディレクティブプログラミング環境

オフローディング対応

RISC-V

- スクラッチから書く新しいアプリケーションへの適用

特定ベンダにロックオンされないオープン標準かつ
マルチプラットフォームなC/C++ベースの開発環境

SYCL

高速動作、リッチなツール群で高効率な開発

課題

高速なGPUでのプロトタイプ化

新規のC/C++コード

エッヂ対応のカスタマイズ

RISC-V

NSITEXE次世代DFP向けSDK

多様なユーザニーズに合わせて
高レベルAPIから低レイヤAPIまでカバー

アクセラレータ基盤

低レイヤ高性能

RVV命令を備える
次世代DFP

アジェンダ

1. 会社紹介
2. RISC-Vと可変長ベクトル命令
3. NSITEXEの次世代RISC-Vプロセッサ向けSDKの構想
4. **何故、SYCLが注目されるのか**
5. まとめ

NSI-TEXE

SYCLの特長

- 標準化団体Khronos Groupが策定するアクセラレータ向け並列化プログラミング言語
  - C++をベースに拡張しており、将来的にはC++標準規格へ統合予定
  - 特定ベンダに依存しないニュートラルな標準規格
- 既存のアクセラレータ向け言語の課題を（部分的に）解決
  - リダクション処理が簡単に記述できる
  - ループ構造がparallel_for記述子により維持される
- 最新規格ではUnified Shared Memoryを使用する事でより簡潔にアプリケーションを記述可能
  - 指示行ベースのプログラミングに近い感覚で、指示行よりも細かな制御まで記述可能

実アプリでのOpenMPとSYCLの比較事例

NSI-TEXE

```
// 3D array
#define U(i,j,k)        u[(i)+((j)+(k)*STRIDE1)*STRIDE0]
#define V(i,j,k)        v[(i)+((j)+(k)*STRIDE1)*STRIDE0]
#define P(i,j,k)        p[(i)+((j)+(k)*STRIDE1)*STRIDE0]
#define Q(i,j,k)        q[(i)+((j)+(k)*STRIDE1)*STRIDE0]

#pragma omp for collapse(2)
    for (kk = 0; kk < DEPTH; kk++) {
        for (jj = 0; jj < HEIGHT; jj++) {
#pragma omp simd
            for (ii = 0; ii < WIDTH; ii++) {
                jp = jj + 1, jm = jj - 1,
                kp = kk + 1, km = kk - 1;

                if (ip > WIDTH  - 1) ip = 0;
                if (jp > HEIGHT - 1) jp = 0;
                if (kp > DEPTH  - 1) kp = 0;
                if (im < 0) im = WIDTH  - 1;
                if (jm < 0) jm = HEIGHT - 1;
                if (km < 0) km = DEPTH  - 1;

                real_t uu   = U(ii,jj,kk),
                       vv   = V(ii,jj,kk);
                real_t uvv  = uu * vv * vv;

                P(ii,jj,kk) = DU * (U(ip,jj,kk) + U(im,jj,kk) +
                                    U(ii,jp,kk) + U(ii,jm,kk) +
                                    U(ii,jj,kp) + U(ii,jj,km) - 6.0 * uu) -
                                                  uvv +  F * (1.0 - uu);
                Q(ii,jj,kk) = DV * (V(ip,jj,kk) + V(im,jj,kk) +
                                    V(ii,jp,kk) + V(ii,jm,kk) +
                                    V(ii,jj,kp) + V(ii,jj,km) - 6.0 * vv) +
                                                  uvv - (F + K)   * vv ;
            }
        }
    }

    return;
}
```

C++ with OpenMP

並列性のあるfor文が
parallel_forに置換わる

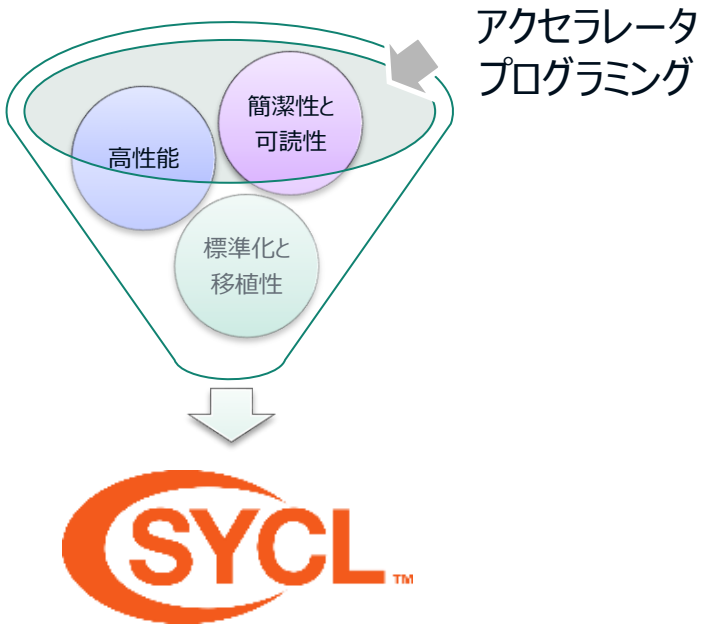
```
// 3D array
#define U(i,j,k)        u[(i)+((j)+(k)*STRIDE1)*STRIDE0]
#define V(i,j,k)        v[(i)+((j)+(k)*STRIDE1)*STRIDE0]
#define P(i,j,k)        p[(i)+((j)+(k)*STRIDE1)*STRIDE0]
#define Q(i,j,k)        q[(i)+((j)+(k)*STRIDE1)*STRIDE0]

queue.parallel_for(
  sycl::range<3>(DEPTH, HEIGHT, WIDTH),
                 [=] (sycl::item<3> item) {
    int ii = item.get_id(2),
        jj = item.get_id(1),
        kk = item.get_id(0);

        if (ip > WIDTH  - 1) ip = 0;
                   T - 1) jp = 0;
                   - 1) kp = 0;
                = WIDTH  - 1;
                = HEIGHT - 1;
                = DEPTH  - 1;

                   J(ii,jj,kk),
          vv   = V(ii,jj,kk);
        real_t uvv  = uu * vv * vv;

        P(ii,jj,kk) = DU * (U(ip,jj,kk) + U(im,jj,kk) +
                            U(ii,jp,kk) + U(ii,jm,kk) +
                            U(ii,jj,kp) + U(ii,jj,km) - 6.0 * uu) -
                                          uvv +  F * (1.0 - uu);
        Q(ii,jj,kk) = DV * (V(ip,jj,kk) + V(im,jj,kk) +
                            V(ii,jp,kk) + V(ii,jm,kk) +
                            V(ii,jj,kp) + V(ii,jj,km) - 6.0 * vv) +
                                          uvv - (F + K)   * vv ;
    });

    return;
}
```

SYCL2020 with USM

SYCLへの期待

SYCL=アクセラレータプログラミング＋
可読性＋高性能＋標準化

- C++標準規格化と連動し、将来的にC++へ統合される計画
- 先進的な機能による高度な並列性の抽出
- 特定のベンダに依らないオープン規格
  - コードの移植性を確保
- ユーザが本質的な処理に集中できる



アジェンダ

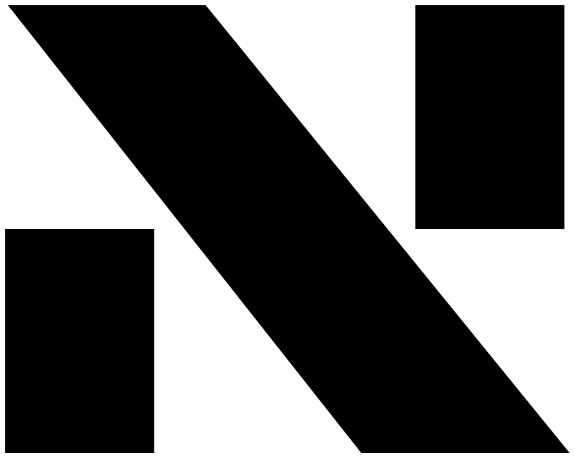
1. 会社紹介
2. RISC-Vと可変長ベクトル命令
3. NSITEXEの次世代RISC-Vプロセッサ向けSDKの構想
4. 何故、SYCLが注目されるのか
5. **まとめ**

まとめ

- RISC-Vで採用される可変長ベクトル命令
  - 既存の固定長SIMDよりも更に進んだ命令セット
    - 演算器の稼働率の向上に主眼が置かれており、より高い実効性能が発揮できる
- RISC-Vの課題
  - 商用IPに比較するとソフトウエアエコシステムがまだまだ未発達
    - 「既存ユーザのコード資産の活用」と「新しいアプリケーションへの対応」の両面での解決が急務
- NSITEXEの次世代RISC-VコアではSDKとしてOpenCL, SYCLを中心とするソフトウエア群を整備し、RISC-Vを取り巻く課題の解決に努めます

Acknowledgment

この成果の一部は、国立研究開発法人新エネルギー・産業技術総合開発機構（ＮＥＤＯ）の委託業務（JPNP16007）の結果得られたものです。



NSI-TEXE