「ネットワークとメディアが拓く新時代-- 感じる･伝える･考える --」2007/11/17

コンピュータの目が危険を知らせる

-超高速/低電力コンピュータへの挑戦-

Copyright (C) 2007 奈良先端大 中島康彦

1

1 最近，衝突防止装置を備えた自動車が登場したことは記憶に新しい．使われているのは，人間の目の役割を果たす２台のカメラ．ステレオ画像処理技術を例に，超高速／低電力コンピュータとはどのようなものであるかについて概観します．キーワードは，スーパースカラ，ＶＬＩＷ，マルチメディア処理専用命令，そして，リコンフィギャラブルプロセッサです．

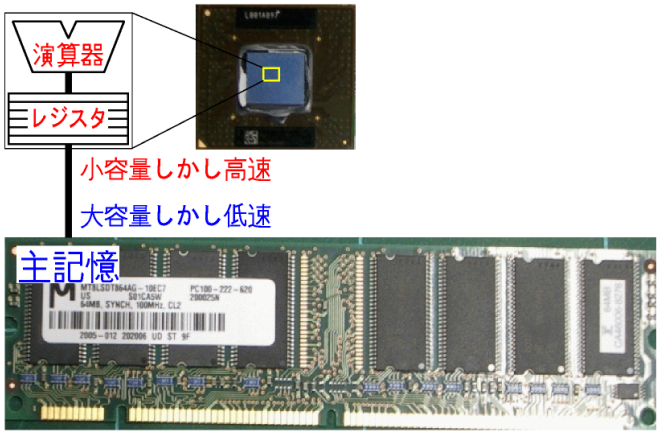
I. はじめに

2

2 まず，どのような世界の話であるかについて，心の準備をしましょう．コンピュータの応用としてのステレオ画像処理を中心に進めていきますが，全体を通じてお伝えしたいことは，決してプログラムの話ではありません．もちろん，最初は，どのような考え方に基づいてプログラムが書かれているかについて，ソフトウェアの観点から一通りの説明をしますが，その後は，ハードウェアの構造と関連付けた話に入っていきます．
最近は，コンピュータが十分に速くなったので，ハードウェアの仕組みを知らなくても，正しく動くプログラムさえ書けば，さほど困ったことにはならないと考える人が増えてきているように感じます．ところが，この数十年間，５年毎に１０倍の勢いで高速化してきたコンピュータは，現在，大変大きな壁にぶつかっています．このことを踏まえて，最近のコンピュータがどのような考え方に基づくハードウェアによって動作しているのかについて，多少なりとも発見されることがありましたら幸いです．

I. 自己紹介

コンピュータの基本構成



3

3 自己紹介を兼ねて，コンピュータのどの部分に関する話であるかについて説明します．スライドは，この数十年間，ほとんど変わっていないコンピュータの基本構造を表しています．上の写真がＣＰＵ（中央処理装置）です．最近のＣＰＵは，膨大な発熱を冷却するための様々な部品に囲まれていて，ＣＰＵ本体を見る機会は滅多にありませんが，どんどん分解していくと，最後にはこのような四角いシリコンの板にたどり着きます．この中には，２進数で加減乗除を行うための回路と，計算のために主記憶から持ってきたデータを一時的に蓄えるレジスタが入っています．ＣＰＵは，主記憶上の「機械命令語」を一つ一つ解釈しながら，加減乗除などの計算を行います．

I. 自己紹介

一般的な機械語命令

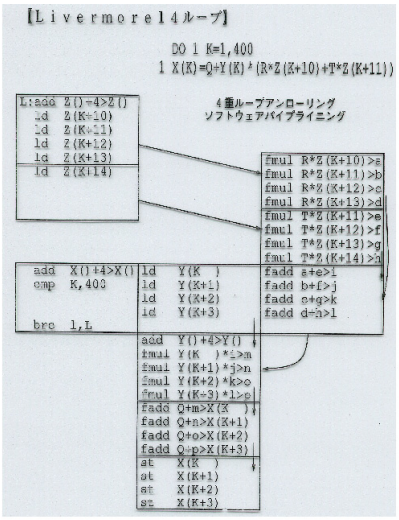
0			31
Load	Reg	Imm./Reg	Reg
Integer	Reg	Imm./Reg	Reg
Store	Reg	Imm./Reg	Reg
Branch	Target Address		

VLIW（Very Long Instruction Word）

0 3	4 5 7	8 37	38 67	68 97	98 127
0	1 0	control, coprocessor (vector etc.)			
0	1 1	float(H)	float(L)	int,l/s	int,l,n,br,j
0	1 2	int,l/s	float(L)	int,l	int,n,br,j
0	1 3	int,l/s	int	int,l	int,n,br,j
0	1 4	int,set,l/s		int,set,l,n,br,call	
0	1 5	float(H)	float(L)	int,set,l/s,br,call	
0	1 6	int,l/s	float(L)	int,set,l/s,br,call	
0	1 7	int,l/s	int	int,set,l/s,br,call	

4 上段は一般的な機械語命令の一例です．上から順に，主記憶からレジスタに「ロード」する命令，「演算」命令，レジスタから主記憶に「ストア」する命令，演算結果に基づいて別の命令に「分岐」する命令です．身の回りの大部分のコンピュータは，このような機械語命令の指示によって動作しています．ところで，機械語命令が昔から変わっていないのに，なぜ５年に１０倍も速くなるのでしょうか．１つは半導体の微細化による動作周波数の向上ですが，これだけでは足りません．もう１つは，単位時間あたりに実行できる機械語命令数の増加です．スーパスカラと呼ばれる方式により，最近では一度に４個の命令を１サイクルに実行することができます．ただし，このような方式を追求してきた結果，消費電力が１００Wを超えるＣＰＵが一般に流通するようになってしまったのです．代わりに最近注目されているのは，下段のＶＬＩWと呼ばれる方式です．１つの機械語命令の中に複数の演算を指示することにより，さほど消費電力を増加させずに性能を向上できます．ただしＶＬＩWには，機械語命令の互換性を維持することが難しいという弱点があります．例えば，最大２演算を同時に指示できる古いＶＬＩWを改良して，最大４演算を指示できるようにするには，機械語命令の形式そのものを変更する必要があり，新しいＶＬＩWにより書かれたプログラムを古いコンピュータで動かすこともできません．

I. VLIWの命令スケジューリング



5 ＶＬＩWを使いこなすには，なるべく多くの演算が同時に実行されるよう，演算を並べなければなりません．この作業を命令スケジューリングと呼びます．ＶＬＩWの場合プログラマが行いますが，スーパスカラの場合ハードウェアがプログラム実行時に行います．実は，この命令スケジューリングに多くの電力が使われます．

I. VLIWとの関わり

20年前 … QA2	学生時代に使わせていただきました
5年後 … VPP500	最初のVLIW型スーパーコンピュータ
?年後 … mGEN	メディアプロセッサ
10年後 … VPP5000	最後のVLIW型スーパーコンピュータ
15年後 … FRV	メディアプロセッサ
20年後 … OROCHI	現在進行形（縁は続くよどこまでも）

6 個人的に関わったＶＬＩW方式のコンピュータです．ＶＬＩWの起源は数十年前に遡ります．１台百億円のスーパコンピュータにも使われましたが，当時は低消費電力ではなく，高性能なものを短期間で開発するという理由で採用されました．当時の技術者の一部は今でもＶＬＩWに関わっています．先日のＨＯＴ−ＣＨＩＰＳでも，各社がＶＬＩWを使ったマルチコアを発表しています．

今日の話題一覧

画像処理を題材に，並列処理と消費電力の関係を概観

覚えて欲しいキーワード

► Energy-Delay積，マルチメディア命令，VLIW

I. VLIWと自己紹介（済）
II. ステレオ画像処理の概要
III. 基礎知識とアルゴリズム
IV. マルチメディア命令と消費電力
V. さらにFPGAによる高速化
VI. 今日の話題のおさらい

7 NAIST

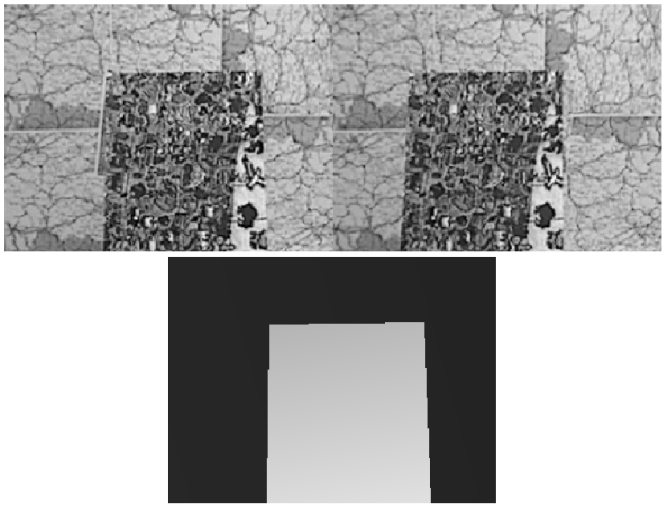
7 これから，ステレオ画像処理を例に用いながら，以上のようなハードウェアの話を進めていきます．ぜひとも覚えておいて欲しいキーワードは，ステレオ画像処理ではなく，「Ｅｎｅｒｇｙ–Ｄｅｌａｙ積」，「マルチメディア命令」，「ＶＬＩＷ」の３つです．

II. ステレオ画像処理の概要

8 NAIST

8 ハードウェアの話はひとまず中断して，ここからしばらくは，ステレオ画像処理の話です．

II–1. 左目画像と右目画像から，奥行き画像を生成



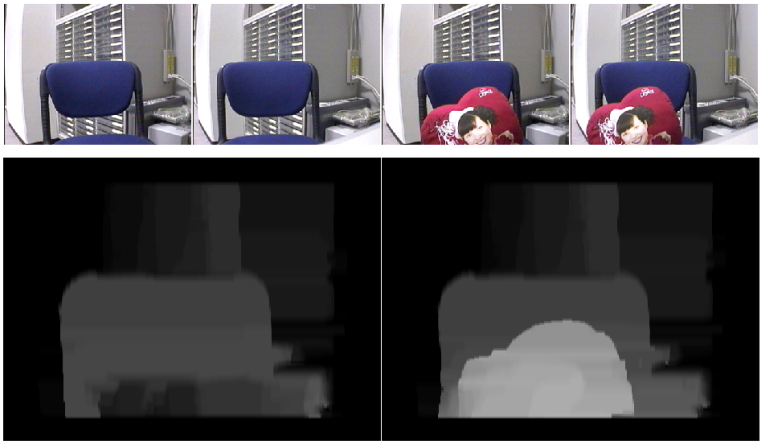
9 NAIST

9 ステレオ画像とは，左目から見た画像を左側に，右目から見た画像を右側にそれぞれ並べた画像です．遠くを眺めておいて，そのままステレオ画像に目を移し，左右の像を重ねることができれば，奥行きのある画像として立体的に見えてきます．ステレオ画像処理とは一般に，この奥行きを計算することを指します．手前に存在する物体ほど明るく表示するのが一般的です．この例では，背景の手前に四角い柱が立っている様子が観察できます．ところで背景と物体の両方に不規則な模様が付けられているのはなぜでしょうか．実は，一般に，模様があるほうが，単純なアルゴリズムでも正確な奥行きを計算できるためです．コンピュータにとって都合の良い画像ということですが，具体的にはあとで説明します．

II-2. 普段はない物体の検知

近距離に出現した物体の検出

▶白領域の出現



10

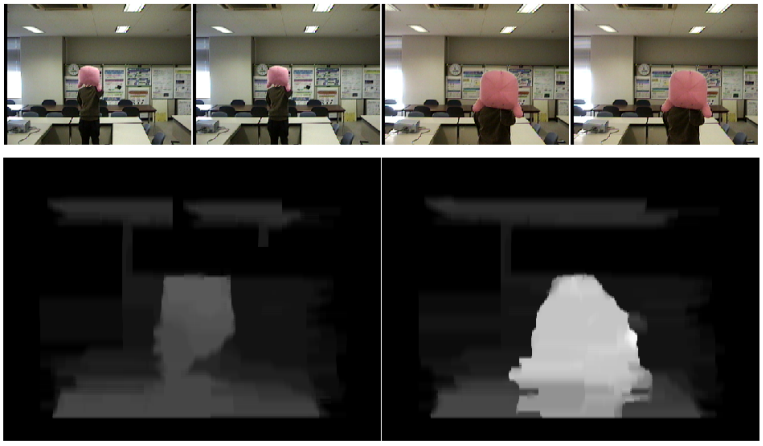
NAIST

１０ 奥行き画像には，多くの使い道があります．普段存在しない物体が急に現れたことを検知するために役立ちます．

II-3. 接近する物体の検知

物体の接近

▶白領域の明度増大



11

NAIST

１１ 画面の一部が徐々に明るく，大きくなってきた場合，何かが接近していると考えられます．衝突予防に使えそうです．

II-4. 画像認識の前処理

物体の切り出しと認識

12

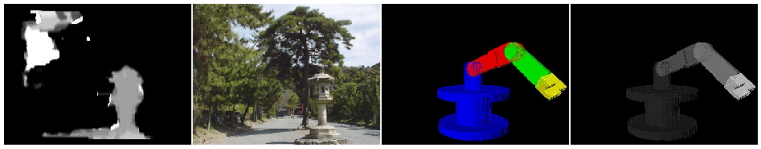
NAIST

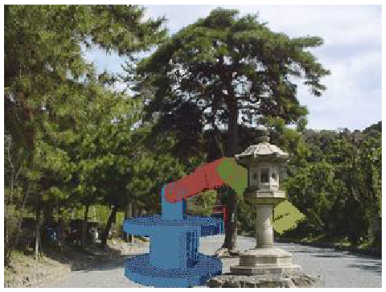
１２ 物体が何であるかを認識するためには，背景を除去する必要があります．奥行き画像を用いて物体を切り抜くことができそうです．

II-5. 実写映像とCGの合成

奥行き情報に基づく合成 (Z-keying)

- 原理的には実写どうしでも可能





13

NAIST

１３ コンピュータグラフィクスは，内部の三次元データから二次元画像を生成します．つまり，最初から奥行き情報を持っています．実写ステレオ画像から得られる奥行き情報と比較し，手前に存在する画像を優先して表示することにより，コンピュータグラフィクスと実写画像を合成することができます．

II-6. 様々な距離測定方法

カーネギーメロン大 金出教授

- 当研究科 デジタルヒューマン学 連携講座
- 1994年頃からVideo-Rate Stereo Machineを開発

TYZX

- 監視カメラによる物体追跡のための視差測定専用プロセッサ

SONY

- レーザー(スリット光)による光切断法

CANESTA

- 点滅光に対する反射光の到着時刻から距離を測定

14

NAIST

１４ ステレオ画像処理では，金出先生が第１人者です．また，画像処理だけによらず，他の様々な方法により，奥行き情報を得る試みがなされています．中には，レーザー光線や点滅光を用いる方法も提案されています．

II-7. アクティブセンサーを使わない利点

アクティブセンサー

- 対象に向けて送出したエネルギーの反射を観測
  電波，レーザー，超音波など
  対象への物理的･心理的影響が大

パッシブセンサー

- 対象が発するエネルギーを観測
  カメラ … 暗闇では役に立たない
  マイク … 音を発しなければ役に立たない
  低消費電力化の可能性
  対象への物理的･心理的影響が小

15

NAIST

１５ ただし，対象物体によっては，電波やレーザーなどを使用することができない場合があります．一方，カメラにも暗闇で役に立たないなどの弱点があります．場合に応じて使い分ける必要がありますが，応用範囲が広く安全なのは，カメラ映像を用いるステレオ画像処理であると考えられます．

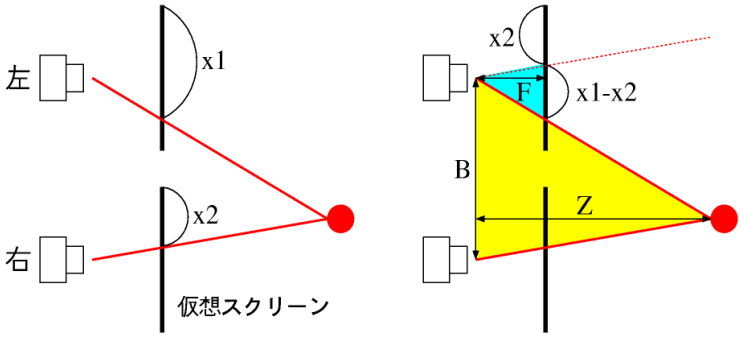
III. 基礎知識とアルゴリズム

16

NAIST

16 ステレオ画像処理の考え方を説明した後，どのようにコンピュータに計算させるかについて話を進めていきます．コンピュータに仕事をさせるための論理的な手順のことをアルゴリズムと呼びます．与えられるデータ形式，および，出力させたいデータ形式を決定し，さらにアルゴリズムを決定すると，プログラムを記述することができます．プログラムを機械語命令に翻訳すれば，あとはコンピュータが手順通りに動作します．記述したプログラムが動作するのにどのくらいの時間を必要とするかは，取り扱うデータ量とアルゴリズムの出来栄えに依存します．

III-1. 複数カメラによる測量



F	仮想スクリーンまでの距離 (固定)
B	カメラ間の距離 (固定)
Z	物体までの距離 (未知) ＝ F*B／d
d=|x1−x2|	仮想スクリーン上の座標差(視差)

17

NAIST

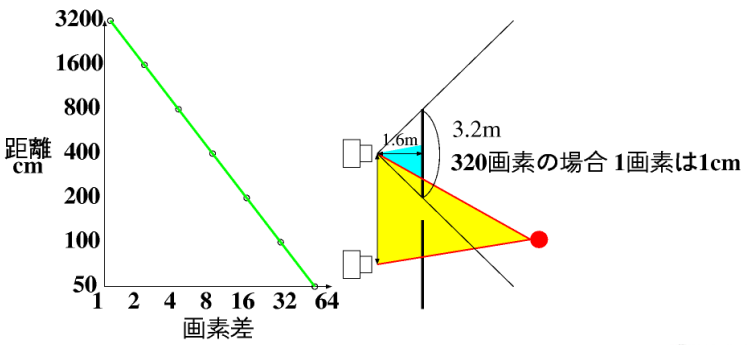
17 カメラと対象物体の関係を真上から見た図です．左右のカメラ画像は，各カメラの前方に配置した仮想的なスクリーン上に投影された画像と考えることができます．ある物体が左カメラ画像では座標ｘ１，右カメラ画像では座標ｘ２に見えたとします．底辺をｄ＝（ｘ１−ｘ２），高さをＦとする三角形と，底辺をＢ，高さをＺとする三角形は相似形なので，Ｚ＝Ｆ＊Ｂ／ｄにより物体までの距離を求めることができます．

III-2. 精度

カメラの解像度=320x240，視野角=90°の場合

▶ F=160cmの場合，d=1画素は160cm先の1cmに相当

カメラ間距離 B=20cmの場合，Z=160*20／d(画素)



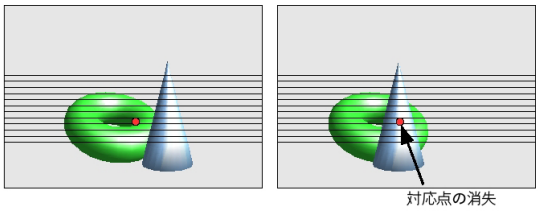
18

NAIST

18 カメラ画像の解像度を３２０×２４０画素，カメラの視野角を９０°とします．この場合，仮想スクリーンまでの距離Ｆを１．６ｍと仮定すると，仮想スクリーンの幅は３．２ｍとなり，１画素あたりのスクリーン上の距離は１ｃｍとなります．カメラ間距離Ｂを２０ｃｍと仮定した場合の画素差と物体までの距離との関係を示したのが左のグラフです．画素差１が奥行き３２ｍ，画素差０が無限遠に相当し，３２ｍと無限遠の間を細かく測定することはできません．Ｂを大きくすれば，より遠くの物体まで細かく測定できますが，逆に近くの物体は認識できなくなります．

III−3. 視差を求めるのは意外に難しい

視差は，物体表面の全ての点ごとに異なる
対応点が存在しないかもしれない

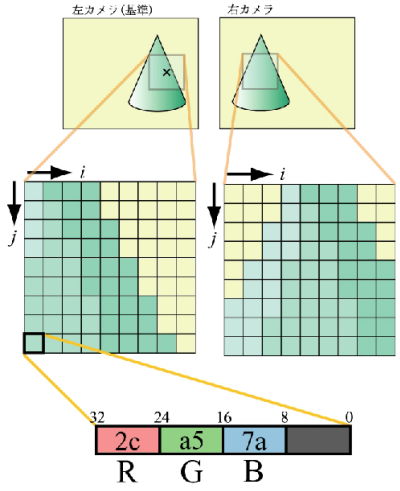


19

NAIST

III−3. 視差を求めるのは意外に難しい（つづき）

左右の画像から，互いに最も類似する一部分を見つけ出す



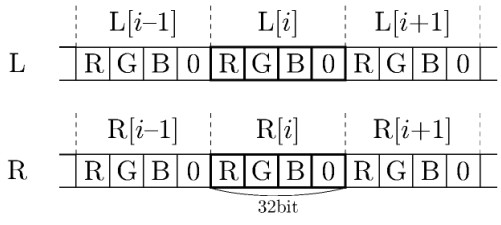
20

NAIST

III−4. 画素差の計算

SAD (Sum of Absolute Difference)…画素差絶対値の総和

- 全体解像度320×240に対し，21×21ピクセル程度のウィンドウ
- 各ピクセルはRGB各8ビット



$$DIFF_i = |L[i]_R - R[i]_R| + |L[i]_G - R[i]_G| + |L[i]_B - R[i]_B|$$

21

NAIST

１９ここまでは簡単な数式により表現できるので，とても簡単でした．しかし，よく考えてみると，対象物体とは何でしょうか．コンピュータが処理しなければならないステレオ画像は，決して真上から見た図ではありません．あくまで正面から見た図ですから，そもそも「物体」がどれかを知ることができません．ステレオ画像処理が少し難しいのは，このためです．左の画像には，ドーナツと円錐が映っていますが，奥行き画像を求める単位は物体ではなく，物体の一部を構成する小さな画像です．あまりに小さな画像を単位として左右の一致点を探そうとすると，そこらじゅうに似たような画像が存在して，正しい一致点を求めることができません．一致点を探す際の小さな画像には，ある程度の大きさが必要と言えます．ところが，左の画像のように，他の物体に隠れて，物体表面の一致点が見えない場合もあるので厄介です．

２０ステレオ画像処理は，一方の画像における部分画像が，他方の画像のどの部分に存在するかをひたすら探す処理であることが，おわかりいただけたと思います．さらに，画像を探すという処理をより具体的に説明していきます．コンピュータが取り扱う画像は，一般に，赤，緑，青の各成分を０から２５５までの階調（１バイト）により表現した１画素分のデータを行方向に並べたものです．左の図では，１画素に４バイトを対応付けています．部分画像を探すと言っても，ある範囲の４バイトの値が全て一致する画像が存在することはまずなく，実際には「最も類似度が高い画像」を探します．

２１部分画像をウィンドウと呼ぶことにします．ウィンドウ間の類似度は，各画素毎に同一色成分の差分の絶対値を合計し，さらに，ウィンドウを構成する全ての画素について合計した「ＳＡＤ」と呼ばれる１つの数値により表現できます．経験的に，２１×２１画素程度の比較的大きなウィンドウを用いてＳＡＤを求めると，ノイズの少ない良好な奥行き画像が得られるようです．もちろん，ウィンドウを大きくするほど計算量が増加し，奥行き画像の生成に要する時間も長くなります．画質と処理時間はトレードオフの関係にあると言えます．

III−4. 画素差の計算（プログラムの一部）

C言語によるウィンドウ間のSAD計算

```
typedef unsigned int Uint;
#define ad(a,b)  ((a)<(b)?(b)-(a):(a)-(b))
#define df(l,r)  (ad((l)>>24&255,(r)>>24&255)
                 +ad((l)>>16&255,(r)>>16&255)
                 +ad((l)>> 8&255,(r)>> 8&255))
Uint L[HT][WD]; 左画像RGB0の配列
Uint R[HT][WD]; 右画像RGB0の配列

wdif(w,lp,rp) Uint w,*lp,*rp;ウィンドウ幅,開始位置
{ int i,j,sad=0;
  for (i=0; i<w; i++) {
    for (j=0; j<w; j++)
      sad += df(*(lp+j),*(rp+j));
    lp += WD;
    rp += WD;
  }
  return(sad);
}
```

22

NAIST

III−5. 計算量の概算

左目画像の1点に対し，SADが最小の右目座標を探索
左目画像1点あたりのad計算回数

▶ RGB各1回 × 縦 × 横 × 探索幅(80画素とすると)

ウィンドウが9×9の場合　3×9×9×80=19440回
ウィンドウが21×21の場合　3×21×21×80=105840回

左目画像の全画面に対しては

ウィンドウが9×9の場合　19440×320×240=1493M回
ウィンドウが21×21の場合　105840×320×240=8129M回

動作周波数1GHzのプロセッサが，毎サイクルad計算を行う場合，

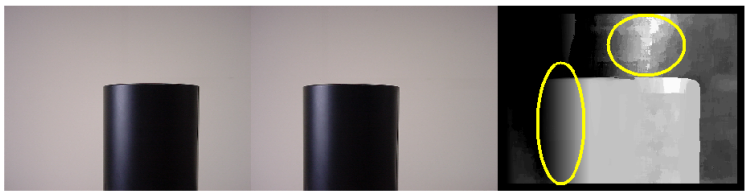
ウィンドウが9×9の場合　1G/1493M=0.7コマ/秒
ウィンドウが21×21の場合　1G/8129M=0.1コマ/秒

23

NAIST

III−6. そのままプログラムを書くと

実験1



物体よりも幅の広い奥行き画像が出る
何もないはずの背景部分にゴーストが出る

なぜこうなるのか考えてみましょう

24

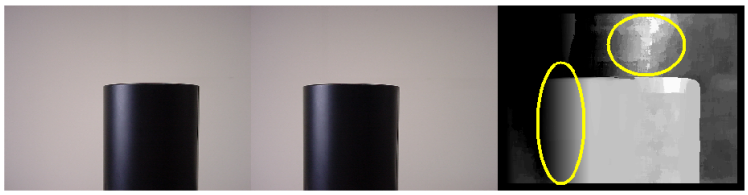
NAIST

２２プログラミング言語をよくご存知の方には，ここまでの説明では物足りないかもしれません．プログラムを初めて見る方には難解ですが，敢えてプログラムの一例を載せておきます．ｗｄｉｆは，ｗ×ｗの大きさの２つのウィンドウのＳＡＤを計算するプログラムです．ｗｄｉｆが内部で使用しているｄｆは，１回実行されると，赤，緑，青からなる１画素のＳＡＤを求めます．さらにｄｆが内部で使用しているａｄは，１回実行されると，１バイトの差分の絶対値を求めます．この段階まで来ると，４に示した機械語命令の数個の組み合わせにより表現が可能となります．顕微鏡の倍率を上げると，より基本的な要素が見えてくるのに似ていますが，実際には，プログラムが動作する機能に達するまで基本要素を組み上げたものがコンピュータなので，未知の物が出てくることは決してありません．ブラックボックスをどこまで開けて理解するかによって，書けるプログラムの質が変わると言えます．

２３３２０×２４０画素の１画面分の奥行き画像を生成するのに，２２のａｄを何回実行しなければならないか概算してみます．左画像のウィンドウ１つに対して，右画像のウィンドウとの類似度計算を横方向に８０回繰り返す場合，ウィンドウが２１×２１画素では，ａｄが約１０万回実行されます．左画像の全体について繰り返すと，約８１億回なので，動作周波数１ＧＨｚのＣＰＵが仮に毎サイクルａｄ計算を実行できたとしても１画面分の距離画像を生成するのに１０秒近くを要します．

２４さて，ステレオ画像処理には膨大な計算が必要であることをご理解いただけたと思います．ここで話題を変えて，実際の計算例をお見せしたいと思います．左画像のウィンドウごとに，右画像を横方向に８０画素分走査して，最も類似度が高い位置との視差を元に，奥行き画像を生成した例が左の図です．それらしく表示されているように見えますが，実は大きく２つの間違いが隠されています．具体的には，丸で囲まれた２箇所が間違っていますが，なぜこのような奥行き画像が生成されるのでしょうか．

III-6. そのままプログラムを書くと(つづき)



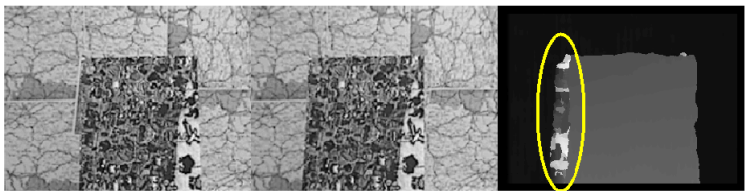
左目画像各点のSAD最小値を表示
明るいほどSADが大

- 背景を「白い物体」として走査している
- 左目画像の背景にあたる部分に模様
- 僅かな濃淡差に影響を受ける

２５ 2つの間違いは，１９に触れたように「対象物体の存在を認識していない」ことに起因します．いずれも左右のウィンドウを比較して，最も類似度が高い位置を視差としているだけですが，上側の間違いは，背景に微妙な明るさの変化があり，たまたま離れた部分との類似度が高かったため，また，左側の間違いは，黒い物体ではなく，白い物体の奥行きとして計算しているために起こっています．

III-6. そのままプログラムを書くと(つづき)

実験2

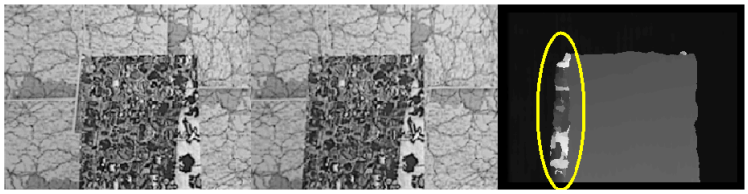


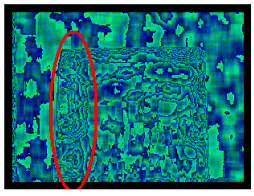
背景と物体に模様があると，綺麗に出る

左端に迷彩模様が出るのはなぜか

２６ 明るさが微妙に変化する模様も何もない大きな領域が間違いを引き起こすのであれば，手っ取り早くきれいな奥行き画像を生成する方法は単純です．左図のように，背景にも物体にも細かな模様をつけることにより，物体であってもなくても類似するウィンドウの位置を正しく計算させ，ある程度正しい奥行き画像を生成することができます．９に説明した「コンピュータにとって都合の良い画像」の一例です．

III-6. そのままプログラムを書くと(つづき)





- 物体左端の近傍に細かな模様
- 右目からは見えない部分に対応
- 対応点が存在しないので混乱
- 無意味な位置でSADが最小になるので白く出る

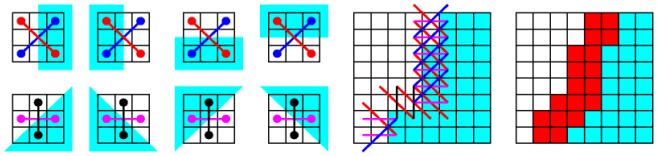
２７ ただし，これだけでは，左側に出ている迷彩模様をなくすことはできません．１９に説明したように「物体表面の一致点が見えない」部分については，たまたま類似度が高いと判定されたランダムな位置が視差となるために，このような模様が現れてしまいます．

III-7. 別の方法

人間の場合（推測）
- ▶画素ごとに距離を掴むのではなく，
- ▶物体の輪郭を認識した上で，左右の像を合わせる

プログラムへの応用
- ▶まず物体の輪郭を調べる
- ▶輪郭を基準にして，遠方から手前に向かって物体を探索

輪郭抽出
- ▶3×3領域の4組の画素差の合計が閾値を超えた場合，輪郭

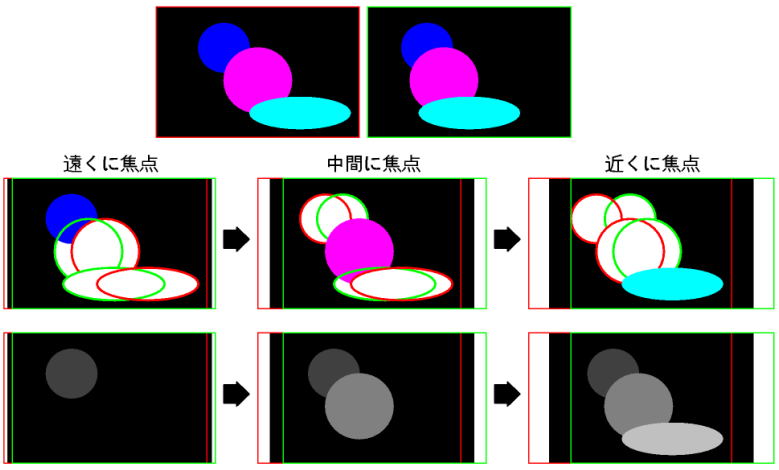


28 NAIST

２８画面全体を走査して物体を認識することができれば，もう少しきれいな奥行き画像を生成できそうです．しかし２３に示したように，今の方法でも１画面分の計算に１０秒近くかかり，これ以上遅くなると応用できる範囲が限られてしまいます．１秒間に何画面も処理できれば様々な応用がありそうです．そこで，まともに物体を認識するのではなく，輪郭情報を元に，せめて「何かある」ことを参考にしてみます．輪郭は，左図のように計算して抽出することができます．

III-7. 別の方法（つづき）

遠方から手前に向かって物体を探索

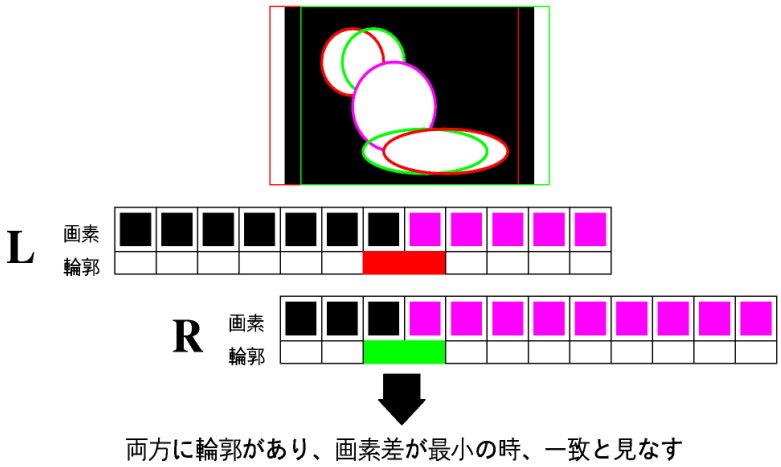


29 NAIST

２９抽出した輪郭情報を用いて，見えない部分の奥行き情報を補う方法を考えてみます．左右の画像を完全に重ねた状態から，少しずつずらせていくと，遠くの物体の輪郭から順に，左右の輪郭が重なっていきます．このことを利用して，遠くにあると思われる「物体らしきもの」から順に奥行き情報を生成し，手前に向かって重ね書きしていきます．このようにすれば，多少の見えない部分を補うことができそうです．

III-7. 別の方法（つづき）

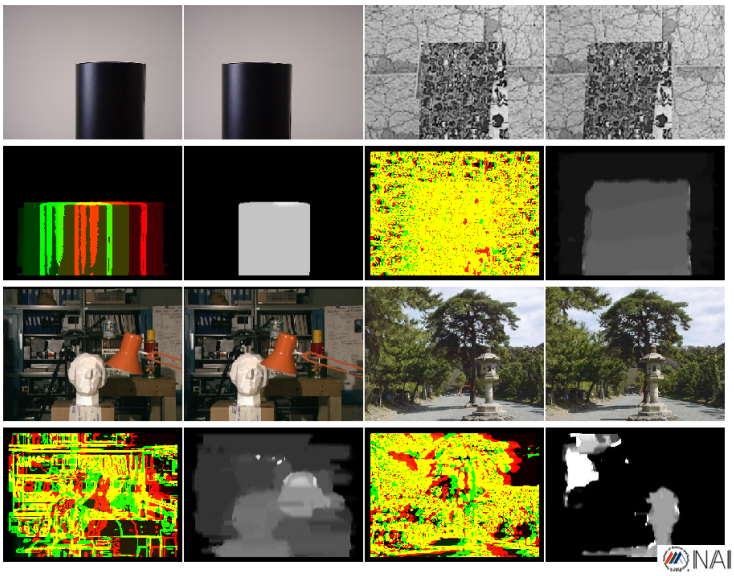
輪郭部分の画素差最小をもって，物体と認識



30 NAIST

３０左図は，輪郭が重なった部分を拡大したものです．左右画像のいずれについても，輪郭の左側が黒，右側が明るくなっています．この状態で類似度が高いと判定される位置は，それほどデタラメにはならないようです．

III-7. 別の方法（つづき）



31

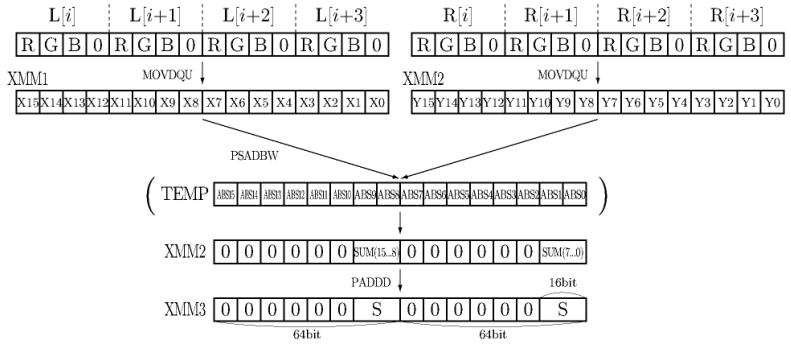
NAIST

IV. マルチメディア命令と消費電力

32

NAIST

IV-1. IntelのSSE2 … Pentium4以降



▶ PSADBW命令が，16バイトに拡張されている

```
MOVDQU L[i],L[i+1],L[i+2],L[i+3]  →  xmm1 (16B)
MOVDQU R[i],R[i+1],R[i+2],R[i+3]  →  xmm2 (16B)
PSADBW xmm1, xmm2                 →  xmm2
PADDD  xmm2, xmm3                 →  xmm3 (16B)
```

33

NAIST

３１ いくつかの画像を使って試すと，かなりまともな奥行き画像が生成できています．さらに別の方法を使うと，よりきれいな画像が生成できるかもしれません．ステレオ画像処理の専門家から見ると相当物足りないでしょうが，話題の中心は，ステレオ画像処理そのものではなくて，高速化と消費電力の関係について概観することなので，アルゴリズムの改良はこのあたりにとどめます．

３２ これまでに説明した計算方法の９５％以上を占めるのが２２のＳＡＤ計算ですが，ＣやＦＯＲＴＲＡＮなどの普通のプログラミング言語を使って書いたプログラムでは，２２のａｄのように１バイトずつしか計算できません．４に示したＶＬＩＷのように，いくつかまとめて一度に計算することができないでしょうか．実はこの目的のために一般的な機械語命令に追加されたのが「マルチメディア命令」と呼ばれる機械語命令です．ＶＬＩＷを導入するには，既存の機械語命令を含めた全面的な作り直しが必要になりますが，命令の追加であれば影響がほとんどありません．ただし，ＶＬＩＷが低消費電力である恩恵は受けられません．あとで，マルチメディア命令とＶＬＩＷを比較します．

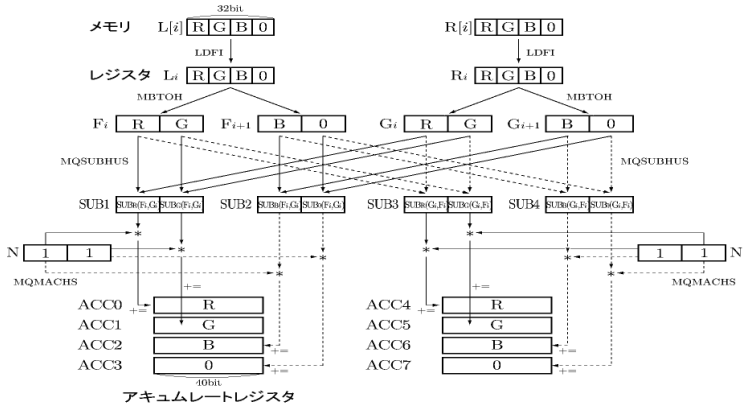
３３ １画素を構成する基本データは，２１に示したように１バイトの大きさです．一方，３に示したレジスタの幅は一般に４から１６バイトです．従来の機械語命令が，レジスタ全体を１つの数値データとして加減乗除する演算を基本とするのに対し，マルチメディア命令は，レジスタを４分割や１６分割して，１バイト程度の演算を一度に複数組実行する仕組みを基本とします．実は，４バイトの加減算を１つ行う回路よりも，１バイトの加減算を４つ同時に行う回路の方が小さくて高速なので，実行速度や消費電力の観点からもマルチメディア命令が有利です．左の図は，Ｉｎｔｅｌが採用しているＳＳＥ２と呼ばれるマルチメディア命令のうちの１つ，ＰＳＡＤＢＷ（何の略かは重要ではありません）を使った計算です．１バイトずつ主記憶からロードして計算するのではなく，一度に１６バイトをロードして計算しています．連続する４画素のＳＡＤを一度に求めることができるので，これだけで１６倍も速くなります．

IV−1. IntelのSSE2 … Pentium4以降（つづき）

```
wdif: ウィンドウサイズ=20×20の場合
    pushl    %ebp
    movl     %esp, %ebp
    movl   12(%ebp),%ecx  /* *lp */
    movl   16(%ebp),%edx  /* *rp */
    pxor     %xmm3,%xmm3
    movdqu  0(%ecx),%xmm1 /* line#1 */
    movdqu  0(%edx),%xmm2
    psadbw   %xmm1,%xmm2
    paddd    %xmm2,%xmm3
    movdqu 16(%ecx),%xmm1
    movdqu 16(%edx),%xmm2
    psadbw   %xmm1,%xmm2
    paddd    %xmm2,%xmm3
    movdqu 32(%ecx),%xmm1
    movdqu 32(%edx),%xmm2
    psadbw   %xmm1,%xmm2
    paddd    %xmm2,%xmm3
    movdqu 48(%ecx),%xmm1
    movdqu 48(%edx),%xmm2
    psadbw   %xmm1,%xmm2
    paddd    %xmm2,%xmm3
    movdqu 64(%ecx),%xmm1
    movdqu 64(%edx),%xmm2
    psadbw   %xmm1,%xmm2
    paddd    %xmm2,%xmm3
    addl     $1280,%ecx
    addl     $1280,%edx
    ⋮
    movdqu  0(%ecx),%xmm1 /* line#20 */
    movdqu  0(%edx),%xmm2
    psadbw   %xmm1,%xmm2
    paddd    %xmm2,%xmm3
    movdqu 16(%ecx),%xmm1
    movdqu 16(%edx),%xmm2
    psadbw   %xmm1,%xmm2
    paddd    %xmm2,%xmm3
    movdqu 32(%ecx),%xmm1
    movdqu 32(%edx),%xmm2
    psadbw   %xmm1,%xmm2
    paddd    %xmm2,%xmm3
    movdqu 48(%ecx),%xmm1
    movdqu 48(%edx),%xmm2
    psadbw   %xmm1,%xmm2
    paddd    %xmm2,%xmm3
    movdqu 64(%ecx),%xmm1
    movdqu 64(%edx),%xmm2
    psadbw   %xmm1,%xmm2
    paddd    %xmm2,%xmm3
    movd     %xmm3,%ecx  /*下位取出し*/
    psrldq   $8,   %xmm3 /*右8Bシフト*/
    movd     %xmm3,%eax  /*上位取出し*/
    addl     %ecx, %eax  /*合算*/
    leave
    ret
```


NAIST

IV−2. FR550 … 8並列VLIW



▶ メディアクワッド飽和付き加算/減算命令

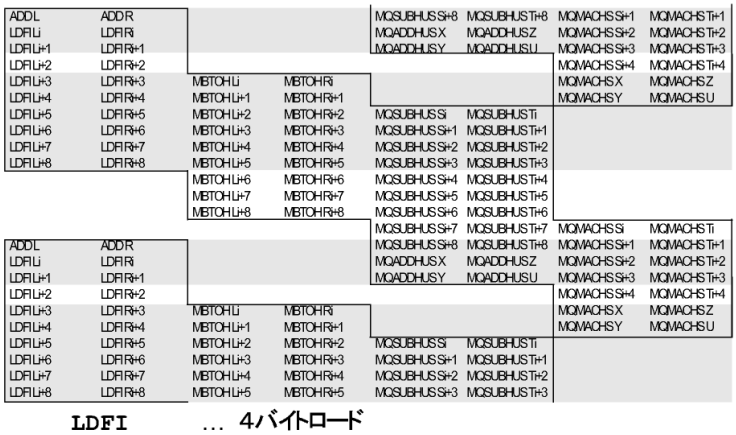
MQADDHUS/MQSUBHUS

飽和減算は減算結果が負の場合0（2命令で同時に8演算）

a−bとb−aの各飽和付き減算結果の和により，絶対値 |a−b|

NAIST

IV−2. FR550 … 8並列VLIW（つづき）



LDFI … 4バイトロード

MBTOH … 各1バイトを2バイトに展開

MQSUBHUS … 2レジスタに対する4つの2バイト飽和付き減算

MQADDHUS … 2レジスタに対する4つの2バイト飽和付き加算

MQMACHS … 2レジスタの4つの2バイト積和演算により累積

NAIST

３４マルチメディア命令を使ったプログラムです．２２にお見せしたプログラムから，このようなマルチメディア命令が自動的に生成できればよいのですが，まだ発展途上なので，今は手作業で書かなければなりません．MOV，MOV，PSADBW，PADDDの４命令を１組として，１６バイト分のＳＡＤ計算を行っています．全体を実行すると２０×２０のウィンドウ間のＳＡＤが１つ求まります．

３５次に，ＶＬＩＷを使ったプログラムについて説明します．１つの横長の機械語命令中に指定できる演算は，従来型の機械語命令とほとんど変わりません．したがって，レジスタを分割使用して，複数画素を同時に計算するマルチメディア命令を追加することは必要です．３４に示した命令列との違いは，予め，多くの演算が同時に実行できるように，命令をスケジューリングしておく点にあります．まずＶＬＩＷにおいて使用する命令を整理します．ＳＳＥ２には，１バイト毎の差分の絶対値を合計する専用命令ＰＳＡＤＢＷがありました．ＶＬＩＷでも同様の命令を追加すればよいのですが，これから使用するＦＲ５５０というＶＬＩＷにはそのような専用命令はありませんので，代わりに，飽和付き減算を使います．飽和付き減算は，A−Bの結果が負になる場合，負とせずに０とする演算です．A−B と B−A の飽和付き減算結果を足し合わせると，差分絶対値が計算できます．ＶＬＩＷでは，２つの減算を同時に実行するよう指定できるので，２ステップで絶対値を求めることができます．ところでＰＳＡＤＢＷは１６個の差分絶対値を毎回集計しますが，ウィンドウ全体のＳＡＤを１６グループに分けたまま累算し，最後に１つにまとめることで，途中の無駄な集計を省くことができます．左図のＶＬＩＷはこのような方法を用いています．少ない命令で表現できることが，必ずしも効率の良さに結び付かない良い例です．

３６左の図は，命令スケジューリング後のＶＬＩＷ命令です．ＦＲ５５０では同時に８個の演算を１命令中に指定することができます．スーパスカラでは，高々４個の命令しか同時実行できませんし，あとで説明するように，消費電力も大きくなりがちです．

IV-3. 消費電力の測定

市販のクランプメータによる測定電流×電圧

37

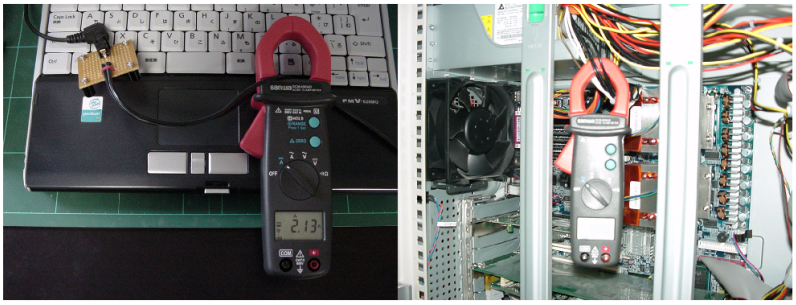
３７実際に消費電力を測定してみます．最近のＣＰＵは消費電力が大きいので，車用のクランプメータを使って測定することができます．

IV-3. 消費電力の測定（つづき）

CPUの近くで測定可能な場所を探す

PentiumM 1.5GHz
16V×2.13A
≒34W

XEON 3.6GHz×2
12V×9.68A
≒116W



38

３８ノートパソコンの内部では，電源供給線を取り出すことが難しいので，ＡＣアダプタの手前で電流を測定します．デスクトップパソコンの場合，ＣＰＵのそばに繋がっている１２Ｖの電源供給線を挟みます．左に示す消費電力は，ＣＰＵがステレオ画像処理を実行している時のものです．プログラムが何も動いていない時の消費電力は，もっと小さくなります．使っていないからと言って，動きの激しいスクリーンセーバなどを動かすと，大型家電並の電力を無駄に消費することになります．

IV-3. 消費電力の測定（つづき）

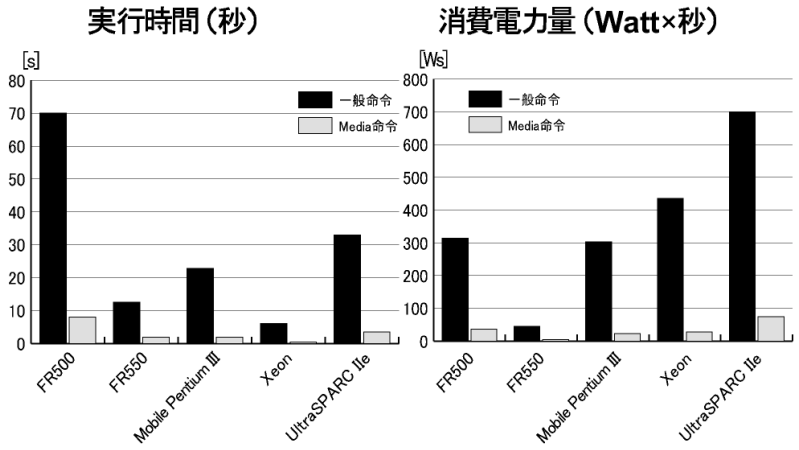
各プロセッサの諸元と性能評価

	（動作周波数）	（測定値）
FR500	240MHz	1.8W
FR550	466MHz	2.2W
Intel Mobile P3	1.0GHz	15W
Intel XEON	2.8GHz	74W
SUN UltraSPARC	500MHz	13W

39

３９いくつかのコンピュータについて，同様の方法を用いて測定した消費電力の一覧です．当時の技術を使って最高の動作周波数を達成した場合，消費電力は動作周波数の３乗に比例すると言われています．

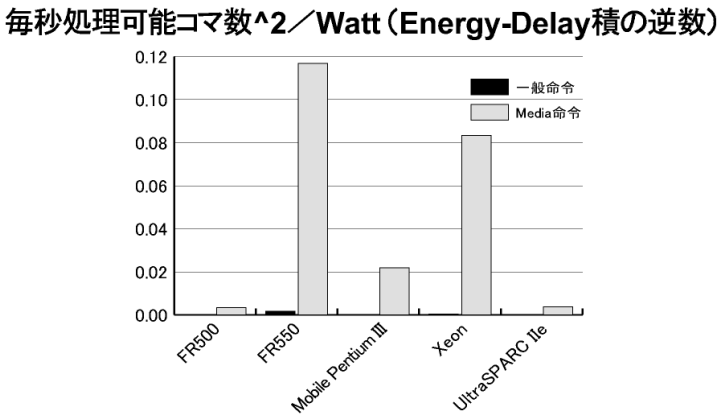
IV-4. 実行時間と消費電力量



40

NAIST

IV-5. 性能と消費電力の両方を考慮する評価尺度



▶エネルギー効率は，FR550 (466MHz) 2.2W が最優秀

41

NAIST

V. さらにFPGAによる高速化

42

NAIST

４０それでは具体的に，ステレオ画像処理に要する実行時間と消費電力との関係を見てみましょう．左側のグラフは１画面分の奥行き画像を生成するのに要した時間，右側のグラフは消費電力量（消費電力×処理時間）です．さらに，濃い棒グラフはマルチメディア命令を用いない２２のプログラムの場合に対応し，薄い棒グラフはマルチメディア命令やＶＬＩＷの命令スケジューリングを駆使した３４や３６のプログラムの場合に対応します．

マルチメディア命令を使うと，単に実行時間が短くなるだけでなく，消費電力量も小さくなることがわかります．また，ＦＲ５５０の消費電力量はほとんど見えないくらい小さいことがわかります．

４１しかし，処理性能があまりに低く，単に消費電力量が小さいだけでは実用的ではありません．このため，処理性能と消費電力の両方を反映する新しい性能指標が必要となります．このような指標の１つがＥＤＰ（Ｅｎｅｒｇｙ–Ｄｅｌａｙ積）と呼ばれるものです．Ｅｎｅｒｇｙは消費電力量，Ｄｅｌａｙは実行時間に対応します．より少ない電力量でより短時間に実行できる優秀なＣＰＵほどＥＤＰ積は小さくなります．ＥＤＰを画像処理の観点からわかりやすく読み替えると，毎秒あたりに処理可能な画面数の２乗をワット数で割ったものになります．詳しくは以下を参考にして下さい．

$$\frac{(\text{総画面数}/\text{実行時間})^2}{\text{電力量}/\text{実行時間}} = \frac{\text{総画面数}^2}{\text{電力量}\times\text{実行時間}}$$

総画面数を１とすると，

1/(電力量 × 実行時間) ＝ ＥＤＰの逆数

左の図は，ＥＤＰの逆数をグラフにしたものです．当時はＦＲ５５０が最優秀であったこと，また，マルチメディア命令を用いることで，いかにＥＤＰが改善されるかがおわかりいただけると思います．

４２さて，ここまでプログラムによる画像処理の一例を示してきましたが，他の考え方に基づいて，劇的に高速化や低電力化を図る方法はないのでしょうか．もちろん，未来の技術ではなく，現時点で使える技術の範囲内でです．

V-1. アーキテクチャ研究用の特注ボード

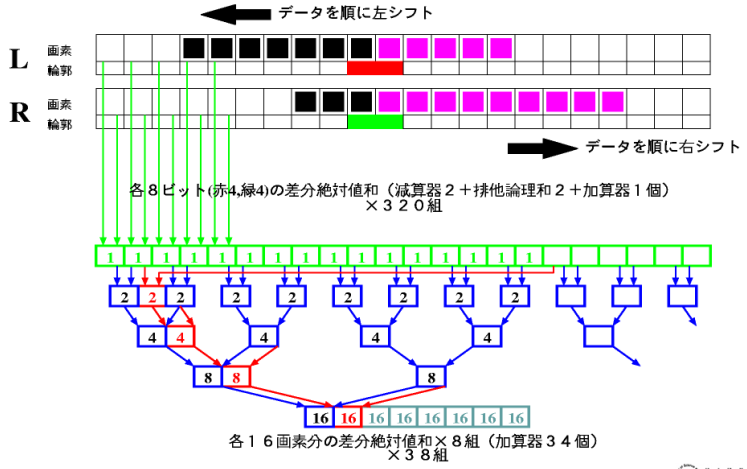
動作周波数は遅いが，膨大な処理を並列実行できる

43

NAIST

V-2. FPGAによるハードウェア化

命令がデータを操るのではなく，データに合わせて演算器を配置
画素メモリと演算器を予め配置してデータを流し込む

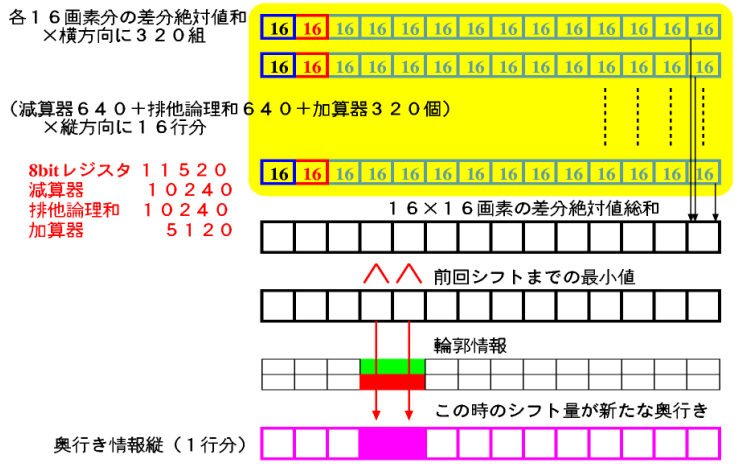


44

NAIST

V-2. FPGAによるハードウェア化（つづき）

16行分のハードウェアを用意



45

NAIST

４３ すでにお気づきの方がいらっしゃると思いますが，画像処理には極めて都合の良い性質があります．４や１６といった控え目な同時実行ではなく，数百から数千の計算を同時にできる可能性があるということです．もちろん，一般に市販されているパソコンに，このような処理をさせることは不可能ですが，多少のお金と知恵があれば，すぐにでもできる方法があります．左の写真は，特別に開発したＦＰＧＡボードと呼ばれるもので，デスクトップ型パソコンに内蔵して使用します．当初は数千万円の開発費を投入しましたが，今では１枚百万円未満で購入できます（著者割引があるかどうかは不明です）．ＦＰＧＡとは Field Programmable Gate Array の略で，プログラムのように記述したハードウェア情報を書き込むことにより，ハードウェアとして使うことができる特殊なＬＳＩです．動作周波数はさほど高くありませんが，多くの演算をハードウェア化することができるので，うまく設計すると，パソコンの何百倍も高速な処理をさせることができます．

４４ ＳＳＥ２では３３のように４画素分のＳＡＤを一度に計算できました．しかし，画像の幅は３２０画素あります．３２０画素のＳＡＤを一気に計算できないでしょうか．実は３０に示した考え方は，１行分のＳＡＤを一気に計算することに相当します．１６×１６のウィンドウのＳＡＤを１つずつ求めるのではなく，横に１画素ずつずれたウィンドウのＳＡＤも同時に求めることを考えます．左の図は，左画像と右画像の１行分から，横に３２０個並べた演算器に一気にデータを供給し，そのまま木構造の演算器を通して，数百のＳＡＤを一度に求める方法を示しています．一般的なＣＰＵの場合，数個の演算器に対してデータをどのような順番で送り込むかという視点でプログラムを書きますが，ＦＰＧＡの場合，データに合わせて，どのように演算器を配置するかという視点で設計を進めます．演算器は膨大にあるので，このようなことが可能になります．

４５ 現在入手可能なＦＰＧＡではまだ困難ですが，近いうちに１６行分の画像データのＳＡＤを一度に求めることができるようになります．この構成では，数万個の演算器が一度に動作することになります．

SSE2は，4画素のSAD計算を1命令で実行(おさらい)
- 2GHzとして，毎秒8,000,000,000画素

FPGAは81920画素(320×16×16)のSAD計算を1サイクルで実行
- 33MHzとして，毎秒2,700,000,000,000画素
- 周波数は60分の1でも，SSE2の320倍の計算能力
  1画面は76,800画素(320×240)
  各画素を40回シフト ⇒ 1コマあたり処理量は3,072,000画素
  理論上は毎秒600コマ(実ハードでも1．6ミリ秒／全画面)

消費電力は数ワット程度
- ただし，汎用プロセッサのような自由度はない

NAIST

VI. 今日の話題のおさらい

NAIST

一般に，消費電力は，動作周波数の3乗に比例
- 低電力化の要望が年々強まっている
- 今後は，プログラム開発の際にも消費電力の考慮が重要

今後の性能評価尺度は，Energy-Delay積

マルチメディア命令やVLIWは低消費電力化に有効

極めて低い動作周波数でも，高性能&低消費電力が可能
- 並列処理のためのプログラミングスキルが必要
- FPGAも1つの選択肢

NAIST

４６ ＳＳＥ２と比較してみましょう．２ＧＨｚで動作するＳＳＥ２が４画素分のＳＡＤ計算を仮に１サイクルで実行できるとします．一方，３３ＭＨｚで動作するＦＰＧＡは８万画素分のＳＡＤ計算を１サイクルで実行できます．動作周波数では６０分の１しかありませんが，単位時間あたりの計算能力では，３２０倍あることがわかります．理論的には，おおざっぱに，毎秒６００画面の奥行き画像を計算できることになります．
もちろん，実際には，パソコンに繋がったカメラから毎秒６００画面のデータを送り込むことは難しいので，実用化するにはいくつものハードルをクリアしなければなりませんが，このような高速化手法もあるということを記憶の片隅にとどめていただければ幸いです．高性能パソコンが安価に購入できる時代に，他に買えるものがないかお困りの方は，このようなＦＰＧＡボードを使ってみてはいかがでしょうか．主にソフトウェアを書くことに興味がある学生に，１人に１台ずつ，このような装置を日常的に使わせることができれば，今までと違った発想ができる学生が育つかもしれません．

４７ まとめます．

４８ 身近な画像処理を例に，最近のコンピュータがどのような構造をしているかについて，主に機械語レベルの話をしました．特に，低消費電力化への要望は年々強まっており，最近でも，国内の電力消費の５％をＩＴ機器が消費していることが取りあげられ，二酸化炭素排出削減のために，ＩＴ機器の消費電力の５０％削減が目標とされる時代になっています．マルチメディア命令やＶＬＩＷが低電力化に有利であることは明らかですが，マルチコアを含めて，これらのハードウエアでは，プログラマが並列度を意識してプログラムを記述することが極めて重要です．普通に高級言語でプログラミングしたものが，これらのハードウェア上で自動的に効率良く実行される日がいずれ来るでしょうが，まだしばらくかかりそうです．

おしまい