大域ディペンダブル情報基盤プロジェクト
～ディペンダブルアーキテクチャグループ　平木研究室～

平木　敬

情報理工学系研究科コンピュータ科学専攻

あらまし　平成１７年度における平木研究室では、平成１６年度に引き続きアーキテクチャ的アプローチによるディペンダビリティの実現、超高速 IDS の実現によるディペンダビリティの確保を目指し（1）CPU そのものを超ディペンダブルとすることを目的とした、FPGA を用いた代用と再構成プロセッサアーキテクチャの研究と、（2）超高速ネットワークを対象とした IDS をアーキテクチャを用いて実現する方式の研究を実施した。FPGA を用いた代用と再構成プロセッサアーキテクチャの研究では、信頼性を失ったことの検出方法を中心に研究開発を実施した。超高速ネットワークを対象とした IDS アーキテクチャでは、マルチストリームの TCP を対象に 40Gbps 以上の速度でストリングマッチングが可能となるように SBT アルゴリズムを拡張し、FPGA を用いて実現した。本稿では、高信頼性プロセッサに関する研究開発について、概要を示す。

１．はじめに

近年ではチップの微細化が進んでいる.回路の複雑性が増大し、テストに必要であるコストが増大しているがチップの限界性能を従来よりも短いサイクルの中で開発することを市場は要求するようになっている.

我々のコンピュータへの依存の割合はさらに高まることが予想され,プロセッサはより高い信頼性を求められている.このため、現在のコンピュータには従来よりも高い耐故障能力も求められている.

本稿ではパイプライン型プロセッサにおける永久故障について議論する.シンプルなパイプラインモデルを保持した状態で数個の耐永久故障能力の付与を目的とする.各パイプラインステージの全てに要素改良技術を使用する必要がある.

単純にユニット数を増加させる手法はパイプラインモデルの変更の必要がないという長所を持つ.ただしハードウェアの単純な追加を続けた場合は面積コストが増大する.

本稿ではパイプラインモデルの実行ステージにおける耐故障性の改良を行う.対象モデル中の実行ステージでは各ファンクショナルユニットが多重化されている.

故障発生時に喪失するファンクショナルユニットの機能を低減するために回路の分割を行う.回路の分割を行った場合にはそれに付随して回路を追加する必要がある.本稿での回路分割対象は論理演算、加算器、乗算器である.これらに対し回路分割を行い,その場合の回路オーバーヘッドを測定する.

２．パイプライン型プロセッサの故障対策

2-1. 永久故障

本稿では故障を永久故障に限定する.一時故障は同一ハードウェア上の再計算により正確な値を得ることが可能である故障である.永久故障は不可能である.永久故障が発生する状況で正常な演算を行うためには,空間もしくは時間冗長を必要である.本稿では空間冗長を利用したモデルをベースに議論する.

本稿はモデルとしてシンプルなパイプラインアーキテクチャに対する永久故障対策について議論する.プロセッサモデルは 6 段パイプライン構成を想定している.

ステージは命令キャッシュアクセス,デコード,レジスタアクセス,演算実行,データキャッシュアクセス,ライトバックである.

同時 4 命令発行を想定しており,デコーダ,演算部位は複数の命令が同時に処理されるものとする.多重化をベースとした故障対策を使用する.回路の内部に存在するデータには十分な冗長が準備されており,それらは常に ECC(Error Checking Code)により保護されている.演算系は全て多重化されており,故障が確認された場合にはその部位を使用しないようにハードウェア上で実装されている.

メモリアクセスユニットについては十分な冗長が準備されており,また内部バス部分まで冗長が準備されており,メモリアクセスに関する故障は発生しないものとする.

故障はシンプルに別ユニットによる 2 回演算を基本にして検出する.故障を検出するための演算ハードウェアは並列に搭載されている.故障検出を行う機構はハードウェア上に実装され,2 回の同一演算は同時に実行/比較される.故障が発生した場合にはファンクショナルユニットが減少する.これにより 2 回演算を実行することが困難になる.この場合は検査側の演算の比率を低下させる.永久故障の場合は故障が永久であるために検出時期を遅らせることが可能である.ただし遅れることにより内部データが破壊され,復帰が困難になる.

稼働可能であるファンクショナルユニット数が減少した場合の具体的な計算/検査比率はそのポリシーに依存する.不正確な演算の検出は 2 回演算/比較により行われる.ある命令が実行ステージにおいて実行失敗した場合にはパイプライン,命令キューのデータはフラッシュされ,その命令のフェッチステージから再開される.ハードウェア故障と判定された場合には内部データが退避されコンフィギュレーションを開始する.

完了した後に再び中断された命令から再開する.もし内部データの退避が不可能であると判定された場合には内部データを全て破棄し,完全なる再起動を行う.

実行するプログラムの途中データはミドルウェアにより適宜セーブされているものとする.これは内部データを破棄する場合でも実行データを保持するためである.

プロセッサのプログラム実行速度性能はその構成要素同士の総合的なバランス調整の結果である.高い性能を得るためには,特定部位の性能向上と同時に構成要素間のバランスを考慮する必要がある.パイプラインモデルの場合には全体性能のスループットが各ステージのスループットを越えることは不可能である.各ステージのボトルネックを評価した上で各種計算資源が割り当てられている.

故障対策においても同様な見方をすることが可能である.性能のボトルネックである部分を解消する立場に基づいた故障対策が必要である.故障による性能低下の比率がパイプラインステージ間に大きな差がある場合には,それぞれのパイプラインの最善手法の統合という選択ではなく,

ボトルネックとなるステージの重点的な解決を行う選択が必要である.

まず、SimpleScalar3.0 を用いてボトルネックステージの特定を行う.設定は表 1 を基本とする.評価ポイントである変数はその場合において変更を行う.

L1 命令/データキャッシュ	4way-64k
最大命令発行数	20M
フェッチ幅	4
デコード幅	4
命令発行幅	4¥
分岐予測	perfect
整数 ALU	8
整数乗算	2
ISA タイプ	alpha

表 1． SimpleScalar の設定

評価に使用するサンプルプログラムは SPEC CINT2000 のうち gcc、mcf、crafty、parser、eon、gap、perlbmk、vortex を使用した.
評価においては次の 3 点を評価した.

1．故障時に発生するキャッシュサイズ変化がプログラム実行速度へ与える影響
2．使用可能であるデコーダ数と実行速度性能
3．使用可能である演算ユニット数と実行速度性能

図 1 はその時点で使用が可能である命令キャッシュの量と実行性能のグラフである.横軸はそれぞれその時点で使用が可能であるキャッシュの状態を示している。横軸の各項は 2 組の数値から成り立っている.前者はキャッシュのブロックサイズを、後者は連想度を意味している.グラフの左端の場合はセット数 1024 バイト,連想度 4 を意味している.左より順に総量の多い順に並べている.縦軸はプログラムの実行性能を表している.それぞれの値は 1024-4 の実行性能により正規化されている.
図 1 では crafty のケースにおいてキャッシュサイズ 512-4 の場合では実行速度は 1024-4 と変わらないがサイズが 1024-1 になると実行速度は 83.4¥%へと低下している.また図 2 の場合も横軸の数値が異なる事以外は同様である.

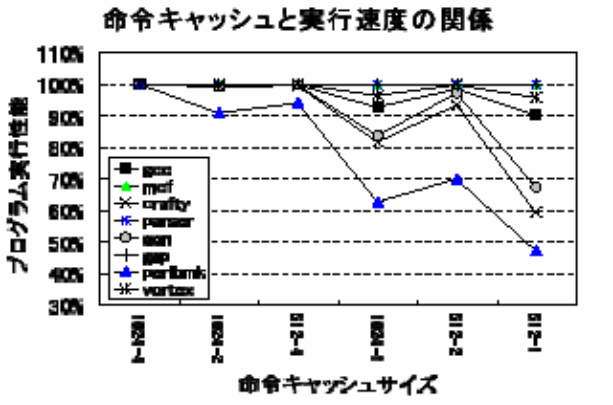

図 1． 命令キャッシュと実行速度の関係

故障がキャッシュメモリに高々数個発生した結果としては以下のことが考えられる.

1．キャッシュメモリ総量が微量減少した
2．連想度が 1 低下した(故障が別ラインに発生)
3．連想度が大幅に低下した(故障が同一ラインに発生)

(1)はグラフの右側へ進む事により読み取ることが可能である.ただし故障により喪失する量はごく微量であり,またグラフの右へ進む事は約半減を意味している.メモリの半減は最悪ケースとして読み取ることが可能である.
(2)は 1024-4 の場合は 1024-2、1024-1 の値の変化のように連想度の微量の低下により読み取ることが出来る.
(3)は 1024-4 から 1024-1 の場合のように,
連想度の大幅な低下により読み取ることができる.ただし、連想度の評価においてグラフに現れているデータは全ての連想度が低下した場合であり,高々数個の故障によりその値まで実行性能が低下するのでは無い.

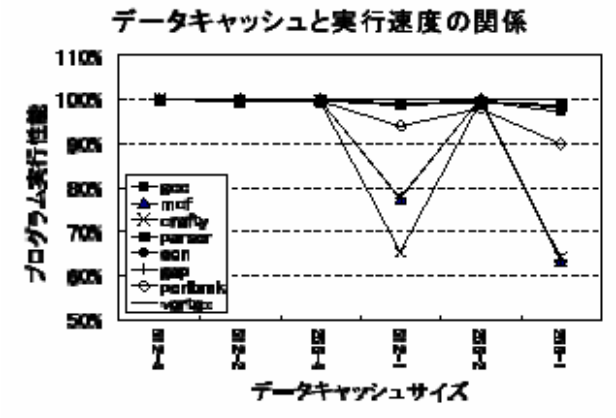

図 2． データキャッシュと実行速度の関係

図 3 はその時点で使用が可能であるデコーダの数と実行性能のグラフである.
横軸はそれぞれその時点で使用が可能であるデコーダの数を示している.
縦軸はプログラムの実行性能を表している.
それぞれの値はデコーダ数 4 の場合の実行性能により正規化されている.
図 3 では parser のケースにおいてデコーダ数 2 の場合では実行速度がデコーダ数 8 の場合と比較して 82%に低下している.図 3 において故障の発生数はその数だけ使用可能であるデコーダの減少数と同等である.よって故障の発生による実行速度性能の変化は図 3 の右側への移行により得ることが出来る.図 3 より,全てのプログラムにおいて
デコーダ数 8 から 4 への減少においては速度性能の低下は見られないが,4 以下になると急激な速度性能低下が発生することがわかる.

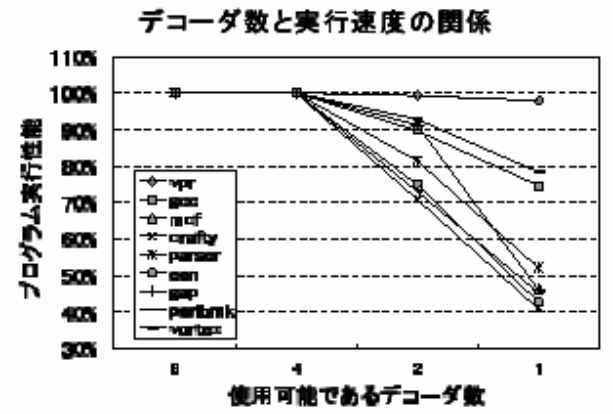

図３．デコーダ数と実行速度の関係

キャッシュにおいてはひとつの故障によりすぐにグラフの右の値へ変化することはなかった.そのために高々数個の故障の発生により大きな速度低下が発生することはないが,デコーダの場合はデコーダ総数が少ないために一つの故障によりグラフの右側にすぐに移動することとなる.つまり故障一つ当りの性能低下の幅が大きいと言える.

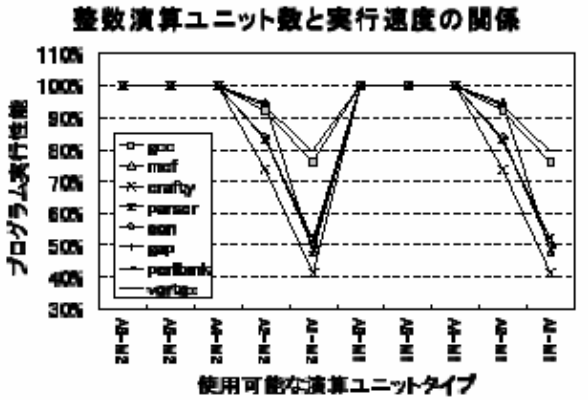

図４．整数演算ユニット数と実行速度の関係

図４はその時点で使用が可能である整数ユニットの数と実効性能のグラフである.横軸はそれぞれその時点で使用が可能である整数演算ユニットの数を示している.横軸の各項は 2 組の数値から成り立っている.A の補助記号のついた前者は整数 ALU の数を,M の補助記号のついた後者は整数乗算演算器を意味している.

グラフの左端の場合は整数演算器 8 個、乗算器 2 個を意味している.縦軸はプログラムの実行性能を表している.それぞれの値は整数 ALU8、整数乗算器 2 の実行性能により正規化されている.

図４では eon のケースにおいて A4-M2 の場合では実行速度は A8-M2 と変わらないが A2-M2 になると実行速度は 84¥%へと低下している.

故障が演算器に数個発生する場合にはその数と同等の演算器が失われることが考えられる.図４では全てのプログラムにおいて ALU 数の減少と共に実行速度性能が低下している.ただし実行速度性能の低下は故障により喪失する機能の数に比例するのではなく,ALU 数が一定ラインを割った場合に性能が特に悪化することが分かる.

これらのデータより故障発生時のデコード,実行ステージの性能低下が大きい.

2-2. ファンクショナルユニットの分割

本稿でモデルとなる実行ステージでは

ロード/ストア命令,論理演算命令,数値演算命令,シフト命令が実行される.ロードストア命令は冗長性と ECC により保護されている.その他の演算は 2 重化により保護されている.このモデルの耐故障性をさらに高める必要がある.

故障による速度低下は一つの故障により重要な回路部位が使用不可能であるためである.重要な回路部位を細粒度化した場合には,故障発生時に機能喪失を部分に押さえることが可能である.故障発生により構成部位を喪失した回路は別の回路が構成要素を喪失した場合に組み合わせて使用する.

実行ステージで処理される代表的な演算としては論理演算,算術演算,ロードストア,浮動小数点演算,大規模演算が挙げられる.ここでは特に論理演算,算術演算について議論する.ロードストア系のデータバスは冗長性により防護されているとの仮定より除外する.またその他大規模演算系は論理演算,算術演算の組合わせにより実行されることより今回は議論から除外する.また算術演算はさらに加算器と乗算器に分ける.これは両者を同一のものとして議論することが困難であるためである.3 種類の回路に対して回路分割について議論する.

回路を分割する場合には他回路で使用可能であることを意識した分割を行う必要がある.密な論理回路を分割方法として回路の各部位にマルチセレクタを挿入する方法とビット幅を縮小する方法がある.ただし前者では複雑な回路にマルチプレクサを挿入することは困難である.またそれを単純なモデルで示すことは困難である.後者ではビット幅縮小の副作用として分割前では要求されなかった演算を要求されるケースがある.このため回路分割を実行するのは困難である.

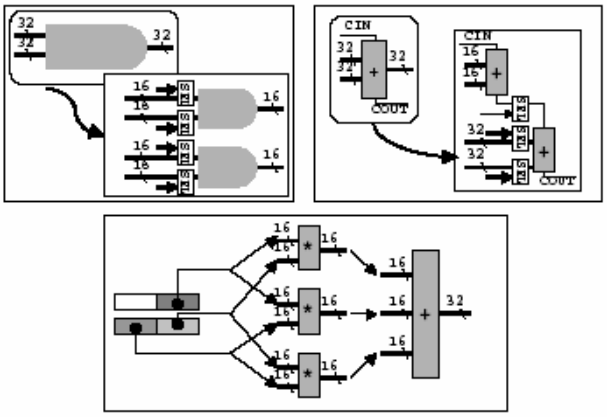

図５．回路分割の模式図

ここでは論理演算の例として AND 演算を使用する.論理演算はそれぞれ 1 ビットずつの入力をそれぞれの論理ゲートを通過させるモデルを仮定する.この場合 AND 演算に他の論理演算を代表させることが可能である.

論理演算は非常にシンプルであるため,その論理演算を分割することは困難である.ここではビット幅を縮小することにより分割する.

図５の左上の模式図はその例である.入力幅 2N の論理演算を入力幅 N の 2 つの演算回路に分割する例である.入力側はその故障状況により配線接続が変化する.実際の論理演算において各ビット演算は互いに独立であるため,

ビット幅縮小による細分化は 1 ビットまで可能である.論理演算においては各ビットの演算回路が等価であるため,そのまま回路の再利用が可能である,このため高い耐故障性がある.

一方で入力サイズが半減した場合と 1bit 幅の場合のどちらのケースでも全ての入力ビットに対して入力データを選択するインターコネクションが必要がある.またより細粒度になるにつれ,入力データの選択のためのデータがより大きくなる.このため,実装において回路サイズの増大する.

加算器のモデルとしてリップルキャリーモデルを使用する.加算器の回路分割はビット分割を使用する.図 5 の右上の模式図はその例である.加算器は分割されている.加算器への入力配線部位には入力データを選択するためのスイッチが接続されている.

入力データはその故障状況に応じて選択され,演算が実行される.加算器では各ビットの演算回路が等価であるため,そのまま回路の再利用が可能である.ただしキャリーが下位ビットから伝搬するためそれぞれの桁の演算が独立でない.このため回路の再利用性を高める場合には非常に大きなインターコネクションが必要となり,高い回路の再利用性を得ることは困難である.

乗算器は次の式により回路分割を行う.

$$A \times B = (A_{up} \times 2^{16} + A_{low})(B_{up} \times 2^{16} + B_{low}) \qquad (1)$$

図 5 はその模式図模式図である.

32bit 乗算を分割された乗算器により実行する.ただし最後にそれに加えて最後に加算器を追加する必要がある.式(1)で現れる項の最上位項は桁よりあふれるためにほぼ無視することが可能である.最後に追加された加算器ではビットシフトを意識する必要がある.ビット幅を分割された乗算器は再利用するためにそれぞれの入力においてインターコネクションが必要である.

今回は実行ステージの数値演算部位の論理演算、加算、乗算についての予備評価を行う.具体的には回路分割を行った際に発生する回路オーバーヘッドを評価する.ロードストアユニット,複雑な演算は対象外とする.これらの評価される演算器は実行ステージにおいて並列に接続されている.演算器への入力されるレジスタデータは既に選択されたと仮定する.その部位の回路は考慮しない.またリオーダに関する回路はより下流で既に実装されたものと仮定する.

故障検出に用いられる数値演算はこれらの演算器で実行されるものとする.ただしどの演算をどの時期に実行するか,数値比較論理回路,具体的な故障検出ポリシーは予備評価に含まないとする.故障検出ポリシーは既に選択され、上流下流でコントロールされているものとする.今回は一種類の演算器に対し 4 個の並列演算器を準備する.これは 2 個を通常実行用、残り 2 個を検出用として使用する方式を基本モデルとするためである.

評価に用いる論理回路として AND 演算を使用する.基本となる AND 演算(AND-def)は 32bit2 入力に対して 32bit を出力する回路を 4 並列したものである.回路分割された AND 演算(AND-div)は入力配線の直後にマルチプレクサをつなぎ,並列配置された AND 演算へつなぐことが可能になった回路である.

厳密には並列配置された 32bit2 入力 AND 演算器 A_i:i=0,1,2,3 において A_i の第 k ビットは A_j(j=0,1,2,3) の第 k ビットと交換可能であるようにスイッチを配置した回路である.

i=j の場合にはマルチプレクサによる配線つなぎ換えは無かったと考えることが出来る.分割された AND 演算においてその接続情報を保持するためのメモリセルは計算に入っていない.

加算器はリップルキャリー型の実装を行う.分割前の加算器回路(ADD-def)は 32bit 2 入力リップルキャリーアダーを 4 並列に並べたものである.回路分割された加算器(ADD-div)は分割前の回路を下位、上位に分割したものを 4 並列にしたものである.分割幅は 16bit である.

分割された 16bit 加算器はそれぞれ上位下位を計算し,出力する.厳密には下位ビットのキャリーを上位へ反映するために上位は下位演算を行った後に受け渡される上位ビットは演算を行うためには下位からのキャリーと演算する数値が必要であるために上位 bit 加算器の入力配線とキャリーにはマルチプレクサが接続されている.

今回の実装では上位演算用加算器同士もしくは下位演算用加算器同士で新規な加算器を構成することを禁止している.これはスイッチ部位の回路量爆発を防ぐためである.このため必ず下位演算用加算器を通過した後上位加算器を通過する構成となっている.

今回は評価を FPGA 上で行う.このため乗算器は搭載されている乗算回路を使用する.今回使用する FPGA では 32bit 乗算回路は存在しないため,MUL-def は存在しない.

今回評価する乗算回路は全て 16bit 分割された乗算器である.16bit 分割されているものの入力信号にスイッチが接続されていない回路 5 は図 5 中の下の回路を実装したものである.

今回回路オーバーヘッドを評価する MUL-div2MUL-div3 は図 5 の乗算器と加算器の間にスイッチを挿入したものである.MUL-div2 は加算器へ出力が可能である乗算器の最大数が 4 であるように接続したもの,MUL-div3 は加算器へ出力が可能である乗算器の最大数が 12 であるように接続したものである.

今回は回路を FPGA 上で実装し,必要な回路量を求める.そのことにより回路分割時の回路オーバーヘッドを測定する.ターゲットとなる回路を VHDL により記述する.記述された VHDL のコンパイルに Xilinx ISE7.1 を使用した.ターゲット FPGA モデルとして Virtex II V1000 を使用した.結果データとしてスライス、LUT(Look up Table)量を得る.1 つのスライスは 2 個の LUT と 2 入力論理演算回路を持つ.LUT は 4 入力 1 出力型である.

	Slice	4-LUT
AND-def	74	128
AND-div	586	896

表 2　AND 回路の必要リソース量

表 2 は AND 回路を通常実装/回路分割型実装した場合の必要リソース量の表である.AND 実行部位を通常実装した場合(AND-def)は Slice 数で換算した場合 74 個,

LUT 数で換算した場合 128 個を使用している.AND 実行部位を回路分割した場合の回路オーバーヘッド量は AND-def と AND-div で使用される回路の差である.そのためこの場合の回路オーバーヘッドは Slice 換算で 512、LUT 換算で 768 である。AND-def では故障が 1 個所発生した場合には 3 ユニットが動作可能であり,4 ユニットに故障が発生した場合には動作が不可能である.

AND-div では故障がベストケースで発生した場合には 32 個の故障が発生しても 3 個のユニットが動作が可能であり,最悪ケースの場合は 4 個の故障で全く動作が不可能

である.ベストケースの耐故障性得るために必要な回路量は 4 入力 LUT 数換算で 128 から 896 へと増大した.

	Slice	4-LUT
ADD-def	64	128
ADD-div	192	320

表 3　ADD 回路の必要リソース量

表 3 は ADD 回路を通常実装/回路分割型実装した場合の必要リソース量の表である.ADD 実行部位を通常実装した場合(ADD-def)は Slice 数で換算した場合 64 個,LUT(Look up Table)数で換算した場合 128 個を使用している.ADD 実行部位を回路分割した場合の回路オーバーヘッド量は Slice 換算で 128、LUT 換算で 768 である.

ADD-def では故障が 1 個所発生した場合には 3 ユニットが動作可能であり,4 ユニットに故障が発生した場合には動作が不可能である.ADD-div では故障がベストケースで発生した場合には 2 個の故障が発生しても 3 個のユニットが動作が可能であり,ワーストケースの場合は 4 個の故障で全く動作が不可能である.ベストケースは上位、下位加算器に交互に故障が発生する場合である.ワーストケースは上位のみ、もしくは下位のみに故障が発生する場合である.ベストケースの耐故障性得るために必要な回路量は 4 入力 LUT 数換算で 128 から 320 へと増大した.

	Slice	4-LUT	18bit-MUL
MUL-div	32	64	12
MUL-div2	416	640	12
MUL-div3	1568	2176	12

表 4　回路分割を行った場合の ADD 演算回路サイズ

表 4 は MUL 回路を通常実装/回路分割型実装した場合の必要リソース量の表である.このデバイスでは 32bit 乗算器が存在せず 18bit 乗算の組み合わせにより実装されているため,分割型実装を基本とした.MUL-div では Slice 数で換算した場合 32 個,LUT(Look up Table)数で換算した場合 64 個を使用している.

回路分割した場合の回路オーバーヘッド量は MUL-div から MUL-div2 へ変更した場合は Slice 換算で 384、LUT 換算で 576 である.また MUL-div から MUL-div3 へ変更した場合は Slice 換算で 1536、LUT 換算で 2112 である.

MUL-div では故障が 1 個所発生した場合には 3 ユニットが動作可能であり,4 ユニットに故障が発生した場合には動作が不可能である.MUL-div2 では故障がベストケースで発生した場合には 4 個の故障が発生しても 3 個のユニットが動作が可能であり,ワーストケースの場合は 4 個の故障で全く動作が不可能である.ベストケースは一つのユニット部位に故障が発生する場合である.ワーストケースは特定部位のみに故障が発生する場合である.

ベストケースの耐故障性得るために必要な回路量は 4 入力 LUT 数換算で 64 から 640 へと増大した.MUL-div3 では故障がベストケースで発生した場合には 4 個の故障が発生しても 3 個のユニットが動作が可能であり,ワーストケースの場合は 4 個の故障で全く動作が不可能である.ワーストケースは加算部位のみに故障が発生する場合である.ベストケースは一つのユニット部位に故障が発生する場合である.

MUL-div2 から MUL-div3 への改良により特定部位への故障の集中するケースへの改良を行った.その改良のために必要な回路量は 4 入力 LUT 数換算で 640 から 2176 へと増加した.

3．考察

論理演算と加算の場合は 1bit モジュールの組み合わせと考えることが可能である.ただしその回路を再利用を実行する場合にはそれぞれの配線にスイッチを挿入する必要がある.そのためその回路本体に比べた場合倍以上の回路量が必要である.

一方で故障が発生した場合に使用不可能になる回路量が減少する場合も存在する.故障発生部位を含めた評価を行う必要がある.また論理演算器と加算器の場合は回路量増加の原因が必ずしも一致しない.論理演算はシンプルな演算かつ各ビットは独立である.演算時におけるビットの独立性は回路量増加幅の低減につながるが演算がシンプルであるために追加するべき回路量が大きい.加算器の場合は演算が多少複雑化したもののビット演算が下位よりのキャリーに依存し,独立性を保持していないために追加回路が増加している.

4．まとめ

H17 年の研究では故障発生時の性能低下率の低減するために演算器に対する細粒度化を行った.回路分割では故障発生部位がベストケースである場合には低減が考えられるもののワーストケースである場合には低減が不可能である.回路分割における回路量増大は 4 並列論理演算器の場合は 4 入力 LUT で 128 から 896,加算器の場合は 128 から 320 へと増加した.乗算器の場合は縮小乗算回路を無視した場合 64 から 640,2112 へと増加することが得られた.

平成17年の外部発表等

■ 受賞等

3 件 （うち国外 3 件、国内該当なし）

1. 平木敬、稲葉真理、玉造潤史、中村誠、菅原豊、他:「SC2004 Bandwidth Challenge, Single Stream, Longest Path, Standard MTU TCP Throughput」, 2004.11.11
2. 平木敬, 稲葉真理、玉造潤史、中村誠、菅原豊、他:「Internet2 Land Speed Record」, 2004.11.9、2004.12.25
3. 平木敬, 稲葉真理、玉造潤史、菅原豊、他:「SC2005 Bandwidth Challenge, Fastest IPv6」, 2005.11.16
4. 平木敬, 稲葉真理、玉造潤史、菅原豊、他:「Internet2 Land Speed Record」, 2005.10.28、2005.10.29、2005.11.10、2005.11.12、2005.11.13 ※5 回更新

■ 主な原著論文による発表 （査読制度のある雑誌への投稿論文に限る）

国内誌(国内英文誌を含む)

1) 亀沢寛之,中村誠,稲葉真理,平木敬,陣崎明,下見淳一郎,来栖竜太郎,中野理,鳥居健一,柳沢敏孝,生田祐吉:「長距離・高バンド幅通信における並列TCPストリーム間の調停の実現」, 先進的計算基盤システムシンポジウム Symposium on Advanced Computing Systems and Infrastructures(SACSIS),(2004)
2) 菅原豊、稲葉真理、平木敬:「インテリジェントNICを用いた高帯域ネットワーク向けTCP通信方式」, 情報処理学会研究報告、2005年並列／分散／協調処理に関する『青森』サマー・ワークショップ(SWoPP2004), OS-97, no.82, 57-64, (2004)
3) 中村誠, 亀沢寛之, 稲葉真理, 平木敬:「協調動作する並列TCPストリームへのPacket Spacingの適用とその評価」, 情報処理学会研究報告、2005年並列／分散／協調処理に関する『青森』サマー・ワークショップ(SWoPP2004), no.81, 199-204,(2004)
4) 平澤将一、平木敬:「プロファイルを利用した値の局所性による高速化手法」, 情報処理学会研究報告、2005年並列／分散／協調処理に関する『青森』サマー・ワークショップ(SWoPP2004), ARC-159, 001-006, (2004)
5) 大平玲、平木敬:「例外依存関係を越える部分冗長性除去」, 情報処理学会論文誌:プログラミング、2005年並列／分散／協調処理に関する『青森』サマー・ワークショップ(SWoPP2004), SIG 1, no.46, 134-148, (2004)
6) 小川周吾、平木敬:「ハードウェア統計情報を用いたプロセスの動的な最適スケジューリング手法」, 情報処理学会研究報告、2005年並列／分散／協調処理に関する『青森』サマー・ワークショップ(SWoPP2004), ARC-159, 055-060, (2004)
7) 下見淳一郎, 河合純, 下國治, 陣崎明, 中村誠, 稲葉真理, 平木敬:「長距離TCP高速化機構の開発」, インターネットコンファレンス 2004,083-091,(2004)
8) 中村誠,菅原豊,玉造潤史,稲葉真理,平木敬:「擬似ネットワーク環境におけるTCP/IPの性能と評価」, IA研究会, 信学技報, vol. 105, no. 219, IA2005-7,001-006, (2005)
9) 菅原豊, 稲葉真理, 平木敬:「細粒度パケット間隔制御の実装と評価」, 2005年並列／分散／協調処理に関する『武雄』サマー・ワークショップ(SwoPP武雄2005), 情報処理学会研究報告 2005, 085-092,(2005)
10) 西川 徹, 稲葉 真理, 平木 敬:「flat-c: 超並列計算機向けC言語の実現」, 2005年並列／分散／協調処理に関する『武雄』サマー・ワークショップ(SWoPP武雄2005), 情報処理学会研究報告 2005,(2005)
11) 石井康雄, 平木敬:「実行パス履歴情報を利用した分岐予測手法」, 2005年並列／分散／協調処理に関する『武雄』サマー・ワークショップ(SWoPP武雄2005) :情報処理学会研究報告 2005, ARC-164, 001-006, (2005)
12) 菅原豊, 稲葉真理, 平木敬:「動的再構成を用いたアプリケーションレイヤ処理エンジンの設計」, デザインガイア 2005,電子情報通信学会技術研究報告 RECONF2005-59～71, 007-012,(2005)

海外

1) Hiroyuki Kamezawa, Makoto Nakamura, Mary Inaba, Kei Hiraki:「Coordination between parallel TCP streams on Long Fat Pipe Network」, 1st International Workshop on Data Processing and Storage Networking: towards Grid Computing (DPSN04),(2004)
2) Tsuyoshi Ito, Mary Inaba:「Theoretical Analysis of Performances of TCP/IP Congestion Control Algorithm with Different Distances」, NETWORKING 2004, Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications: Third International IFIP-TC6 Networking Conference, LNCS 3042, 962-973, (2004)
3) Makoto Nakamura, Hiroyuki Kamezawa, Junji Tamatsukuri, Mary Inaba, Kei Hiraki, Kenji Mizuguchi, Kenichi Torii, Satoru Nakano, Shoichi Yoshita, Ryutaro Kurusu, Masakazu Sakamoto, Yuki Furukawa:「Long Fat Pipe Congestion Control for Multi-Stream Data Transfer」, Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN2004),294-299, (2004)
4) Hiroyuki Kamezawa, Makoto Nakamura, Mary Inaba, Kei Hiraki:「Coordination between parallel TCP streams on Long Fat Pipe Network」, 1st International Workshop on Data Processing and Storage Networking: towards Grid Computing (DPSN04),041-048,(2004)
5) Yutaka Sugawara, Kei Hiraki:「A computer architecture education curriculum through the design and implementation of original processors using FPGAs」, Proc. of Workshop on Computer Architecture Education (WCAE 2004),003-007, (2004)
6) Yutaka Sugawara, Mary Inaba, Kei Hiraki:「Over 10Gbps String Matching Mechanism for Multi-Stream Packet Scanning Systems」, Field-Programmable Logic and Applications, 14th International Conference (FPL 2004), LNCS 3203,484-493,(2004)

7) Rei Odaira, Kei Hiraki:「Partial Value Number Redundancy Elimination」, Proceedings of the International Workshop on Languages and Compilers for Parallel Computing (LCPC 2004), Lecture Notes in Computer Science,(2004)
8) Hiroyuki Kamezawa, Makoto Nakamura, Junji Tamatsukuri, Nao Aoshima, Mary Inaba, Kei Hiraki, Junichiro Shitami, Akira Jinzaki, Ryutaro Kurusu, Masakazu Sakamoto, and Yukichi Ikuta:「Inter-layer coordination for parallel TCP streams on Long Fat pipe Networks」, Super Computing 2004, High Performance Networking and Computing (SC2004),CD-ROM,(2004)
9) Shoichi Hirasawa, Kei Hiraki:「Utilizing Dynamic Data Value Localities in Internal Variables」, 5th International Conference, Parallel and Distributed Computing: Applications and Technologies(PDCAT), LNCS 3320,305-309, (2004)
10) Makoto Nakamura, Ryutaro Kurus, Felix Marti, Masakazu Sakamoto, yukichi Ikuta, Junji Tamatsukuri, Yutaka Sugawara, Nao Aoshima, Mary Inaba, Kei Hiraki:「Experimental Results of inter-layer cooperative hardware for TRC-TCP on 10Gbps Ethernet WANPHY 18,000km Network」, Third International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet), (2005)
11) Rei Odaira, Kei Hiraki:「Sentinel {PRE}: Hoisting beyond Exception Dependency with Dynamic Deoptimization」, Proceedings of the 2005 International Symposium on Code Generation and Optimization (CGO),328-338,(2005)
12) Shoichi Hirasawa, Kei Hiraki:「Software SMLP Execution with Side-Effect Slice Exclusion」, Proceedings of the 2005 International Conference on Parallel and Distributed Processing Techniques and Applications(PDPTA '05),vol.2, 832-838, (2005)
13) Yutaka Sugawara, Mary Inaba, and Kei Hiraki:「High-speed and Memory Efficient TCP Stream Scanning using FPGA」,Proc. of 15th International Conference on Field Programmable Logic and Applications (FPL2005),45-50,(2005)
14) Junji Tamatsukuri, Katsushi Inagai, Mary Inaba, and Kei Hiraki:「Experimental Results of TCP/IP data transfer on 10Gbps IPv6 Network」, Proc. of Fourth International Workshop on Protocols for Fast Long-Distance Networks(PFLDnet 2006), (2006)