Vol.2018-ARC-232 No.14
2018/7/31

プログラムフェーズを利用した最適キャッシュ管理パラメータの再現手法

渋江 陽人[†1,a)]　野村 隼人[†1]　入江 英嗣[†1]　坂井 修一[†1]

概要：プロセッサの性能向上を目指し，キャッシュ・マネジメントにおいても様々なアルゴリズムが提案されてきた．近年では，特定の挙動を想定せず，プログラムの挙動を用いて学習を行うことで汎用的に高性能となるような手法が数々提案されている．アクセスの予測器 [1] やシグネチャ [2] など様々な手法がある中で，最適なパラメータを学習し，十分な性能が引き出せるようになるまで時間がかかってしまうことも少なくない．あまり長い時間がかかってしまうと，学習がある程度できた時点で既にプログラムの挙動が変化し，学習結果が有効ではなくなってしまっている，という危険性が考えられる．そこでフェーズの概念に注目し，プログラムの挙動が変化しても継続的に細粒度の学習が行える機構を提案する．簡単な評価として，ストリームプリフェッチャのパラメータに適用したところ，いくつかのベンチマークにおいて性能向上が見られ，フェーズを考慮することの有効性が示唆された．

Shibue Akito[†1,a)]　Nomura Hayato[†1]　Irie Hidetsugu[†1]　Sakai Shuichi[†1]

1. はじめに

プロセッサの処理速度を改善する上で，データアクセスの高速化を目的にキャッシュ・メモリが用いられている．近年ではキャッシュを階層的に配置した構成が用いられるだけでなく，効率的にキャッシュを使用できるように様々な手法が提案されている．データ置換アルゴリズム [1], [3] やプリフェッチ [4] といった技術では，利用するデータがキャッシュに存在するよう，限られた容量しかないキャッシュにどのデータを残すかを判断し，近い将来に使用されると推測されるデータを予めキャッシュに格納することができる．無数のデータアクセスパターンが考えられるため，生じる可能性の高い特定のパターンに注目した手法 [3] や，実際に生じたアクセスパターンを学習しながら次のアクセスを予測する手法 [1], [2] が研究されている．

一方で，プログラムの特徴やプログラム中での繰り返しを検出する，フェーズ検出と呼ばれる分野が存在する [5], [6], [7], [8]．最適化のために複雑な学習を行うような場合，十分な学習が行われるまでに長い時間がかかってしまうという問題が起こりうる．

そこで本論文では，フェーズの概念を元に，現在のフェーズが過去にも現れていた場合，その過去の時点のパラメータを復元することを考える．すなわち，プログラム全体を通して最適なパラメータを探索するのではなく，小さく分割した部分ごとに最適なパラメータを学習する．また，実際にプリフェッチャに簡単な機構を追加し，効果を検証する．

本論文の構成は以下の通りである．2 章でハードウェアによるプリフェッチについて述べる．3 章でフェーズの概念と検出手法について説明し，4 章でそれを用いたプリフェッチャの改良について説明する．5 章で実験環境，6 章で改良した結果を述べ，7 章で考察，8 章でまとめを行う．

2. ハードウェアプリフェッチ

将来必要になるデータをハードウェアが予測し，下位のキャッシュやメインメモリから対象となるキャッシュに予め読み込んでおく技術がハードウェアプリフェッチである．配列として連続領域に配置されたデータを順番に調べるような場合に効果的な手法が，ストリームプリフェッチ [4] である．

ストリームプリフェッチャは次のように動作する．動作する様子を図 1 に示す．まずキャッシュミスが発生したとき，そのアドレスをテーブルに保管する．テーブルに保管されているアドレスの前後の一定範囲のアドレスを監視す

[†1] 現在，東京大学
Presently with The University of Tokyo
[a)] shibue@mtl.t.u-tokyo.ac.jp

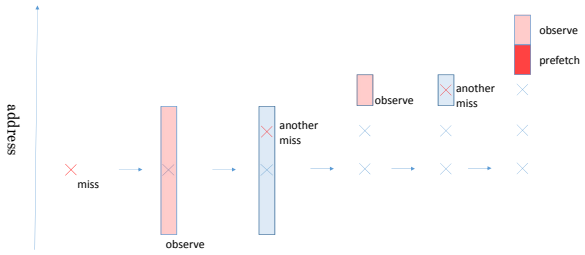

図 1　steps of stream prefetcher

る．監視している範囲でキャッシュミスが発生したとき，保管していたアドレスから見て新たにミスしたアドレスの延長線上にある一定範囲のアドレスを監視するように範囲を切り替える．切り替えた後の範囲でキャッシュミスが発生したとき，監視範囲の延長線上にある一定範囲をキャッシュに読み込み，読み込んだ範囲のさらに延長線上にある一定範囲を監視するように範囲を切り替える．以降，監視範囲にミスがあるごとにキャッシュへの読み込み（プリフェッチ）と監視範囲の更新を繰り返す．パラメータとして，監視する範囲，プリフェッチする量がある．

具体例を挙げる．監視する範囲は 32 バイト，プリフェッチ量は 64 バイトとする．アドレス 1000 にてキャッシュミスが発生したとする．このとき，テーブルに 1000 が登録され，前後 32 バイト，すなわち 968 から 1032 が監視される．次にアドレス 1016 にてキャッシュミスが発生したとする．このとき，監視範囲が 1017 から 1048 に切り替えられる．次にアドレス 1032 にてキャッシュミスが発生したとする．このときアドレス 1033 から 64 バイト，1096 までがプリフェッチされ，監視範囲が 1097 から 1128 に切り替えられる．

実際には，プリフェッチにかかる時間を考慮し，プリフェッチが間に合わないと考えられる，キャッシュミスの直後のアドレスをある程度スキップしてプリフェッチが行われる．また，改良として，プリフェッチする量を更新のたびに大きくする手法 [9], [10]，キャッシュミスに限らず任意のメモリアクセスにおいてプリフェッチをトリガする手法 [4] などが存在する．以下では簡単のため，プリフェッチを行いテーブルの対応するエントリを更新（監視する範囲の更新やプリフェッチ量の更新）することを，単にストリームの更新と呼ぶこととする．

3. フェーズ検出

3.1 概要

ひとつのプログラムでも，実行中の短い期間ごとに注目すると，ある期間ではシーケンシャルアクセスが多く行われ，ある期間ではランダムアクセスが多く行われる，というように処理が変化していることが分かる．逆に，ある期間でアクセスしたデータが別の期間でも同じようにアクセスされている，というように似通った処理が時間をおいて行われることもある．そこで，一定期間ごとに処理の特徴を抽出し，処理の変化を検出したり，特徴ごとに分類したり，ということができる．これをフェーズ検出と呼び，一定の特徴が持続する連続した期間をまとめて，ひとつのフェーズと呼ぶ．

フェーズ検出は 2003 年に IBM の Michael J. Hind らによって提唱され [5]，プログラムの解析や最適化の上でたびたび用いられてきた．フェーズは漠然と定義され，どの程度の長さの期間で捉えるか，どのような特徴に注目するかによって異なる検出結果が得られる．そのため，検出されたフェーズをどう利用するかを考え，適切な検出手法を用いることが求められる．

3.2 BBV

広く用いられてきた手法に Basic Block Vector（BBV）がある [6], [7], [8]．ベーシックブロックとは，プログラム中の，分岐や合流を含まない連続した命令列のことである．命令実行ごとに，実行された命令が属するベーシックブロックに対応するカウンタをインクリメントする．一定時間経過後にカウントを出力し，以前の出力と比較して一定値以上の差が見られたときに動作の変化を検出する．それぞれ，命令数として 10M 命令，比較の方法としてマンハッタン距離，比較の閾値として 2 期間の合計カウント数の 4 %，すなわち 800K が一般に用いられる．

3.3 2 次ストライド

シーケンシャルアクセスが見られるとき，アクセスされた個々のアドレスの間隔はストライドと呼ばれる．例えば，1000，1008，1016，1024，……というアクセスが見られたとき，このストライドは 8 である．また，シーケンシャルアクセスが複数見られるとき，その先頭アドレス同士の間隔を 2 次ストライドと呼び，これをフェーズ検出に用いることを提案している [11]．例えば，1000，1008，1016，1024，1100，1108，1116，1124，というアクセスが見られたとき，1000，1008，1016，1024 という系列と 1100，1108，1116，1124 という系列で捉えることができ，2 次ストライドとして 1000 と 1100 の差分の 100 を考えることができる．

一定数だけ命令を実行している間に，2 次ストライドをインデックスとしてカウントを行う．例えば期間の長さとして 10M 命令を採用した場合，10M 命令ごとに 10, 20, 30, ... や 100, 0, 50, ... といったカウントが得られる．カウンタは二次元になるが，以下では単にカウンタと言ったとき，ある期間に対応する一次元のカウンタのことを指す．

この例では 100 のカウントが 1 だけされることになる．例に加えて 1200，1208，……というアクセスが行われた場合，100 のカウントが 2 となる．

一定命令数だけ実行した後，現在のカウンタと以前のカウンタを比較してマンハッタン距離を計算し，距離が比較している 2 カウンタの合計カウント数の一定割合以下だった場合に，2 つの期間は同じフェーズであると見做す．過去のカウンタのいずれとも類似していない，つまり過去のカウンタの全てに対して距離が閾値以上だったとき，新しいフェーズと見做す．

フェーズを代表するカウンタとして，特定のカウンタを採用する方法と，同じフェーズに分類されたカウンタの平均を取る方法が考えられる．前者の場合，フェーズの境目となるような遷移期間が代表に選ばれた場合，不要なフェーズが検出される可能性がある．一方後者の場合，時間経過に沿って少しずつカウンタが変化しているようなときに全て同じフェーズと見做される可能性がある．不要なフェーズが検出されたとしても，十分長い時間が経過した後ではほぼ同じ学習結果となっていると考え，本論文では前者の，初めてフェーズが検出されたときのカウンタを採用する手法を用いることとする．学習の詳細については 4 節で述べる．

3.4 その他のフェーズ検出

近年の手法に，メモリを中心に捉えた手法がある [12]．メモリフェーズと呼ばれるこの手法では，ページごとのアクセス回数を記録し，そのカウンタを比較する．マルチコア環境においてスクラッチパッドメモリを効率良く利用するため，どう配分するかといった制御に有効であることが示されている．

4. ストリームプリフェッチャの改良

2 章で述べたストリームプリフェッチャのうち，ストリームの更新ごとにプリフェッチ量を増加させるものを対象として性能の向上を図る．ここでは，追跡しているストリームごとに個別にプリフェッチ量が定まっており，ストリームの更新の際，対応するプリフェッチ量を倍にするものを特に取り扱う．また，プリフェッチ量については上限を定める．

フェーズの種類ごとにどの程度の長さのプリフェッチが有効であるか記録する．すなわち，ストリームプリフェッチャのテーブルについて，テーブルからの追い出し時に最終的なプリフェッチ量を出力し，プリフェッチ量ごとにカウントする．カウントの合計の半分以上カウントされていた最も大きなプリフェッチ量を，最も有効なプリフェッチ量と見做す．ただし，大きいプリフェッチ量に対応するカウンタは小さいプリフェッチ量のカウントを内包するため，大きいプリフェッチ量からの累積値で考える．例えばある種類のフェーズについて，プリフェッチ量が 16 バイトのストリームが 10 個，32 バイトのストリームが 100 個，64 バイトのストリームが 30 個見られたとき，合計カウント数は 140 であり，64 バイト以上に対するカウントは 30，32 バイト以上に対するカウントは 130 となるため，最も有効なプリフェッチ量は 32 バイトと見做される．

同種のフェーズでは同種のアクセスパターンが生じるものと考え，以前最も有効だったプリフェッチ量を考慮してプリフェッチ量の初期値を定める．プリフェッチ量が上限で固定されないよう，最も有効だったプリフェッチ量の半分を初期値に採用する．この例の場合，同じフェーズに分類された期間ではプリフェッチ量を 32 バイトとしてプリフェッチを始める．

一定命令数を実行し，カウンタが完成するまでは現在がどのフェーズであるか判断ができないため，直前の期間と同じフェーズが持続するものと仮定して処理を行う．この点については，フェーズの出現パターンからの現在のフェーズの予測が可能である．7 章にて議論する．

表 1 Parameters

L1D cache	size	32 KB
	Latency	3 cycles
L1I cache	size	32 KB
	Latency	3 cycles
L2 cache	size	256 KB
	Latency	15 cycles
L3 cache	size	4 MB
	Latency	30 cycles
prefetcher	initial degree	16 Byte
	max degree	256 Byte
	table size	16 entry
	distance	16 Byte
phase detector (BBV)	threshold	4%
	interval	10M
	counter table size	infinite
	BBV table size	64 entry
phase detector (HD-stride)	threshold	6%
	interval	10M
	counter table size	infinite
	HD-stride table size	64 entry

5. 実験環境

シミュレータとして，鬼斬弐 [13] を用いる．プリフェッチャにフェーズ検出器を追加している．フェーズ検出器には BBV を用いたものと 2 次ストライドを用いたものの 2 種類を用意し，それぞれ別々に実験を行う．パラメータは表 1 に示す．ベンチマークとして，SPEC CPU2006 を使用する．ベンチマークはそれぞれ，始めの 10G 命令をス

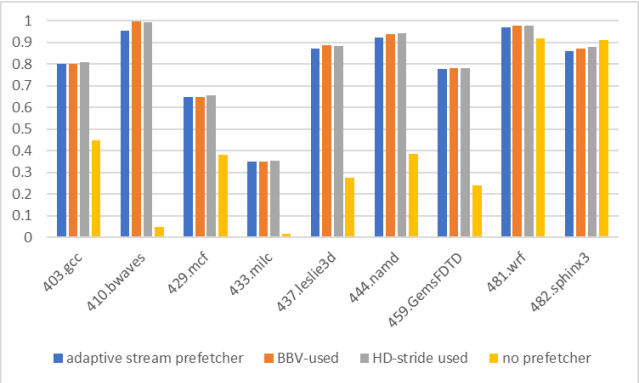

図 2　read hit rate on L3 cache

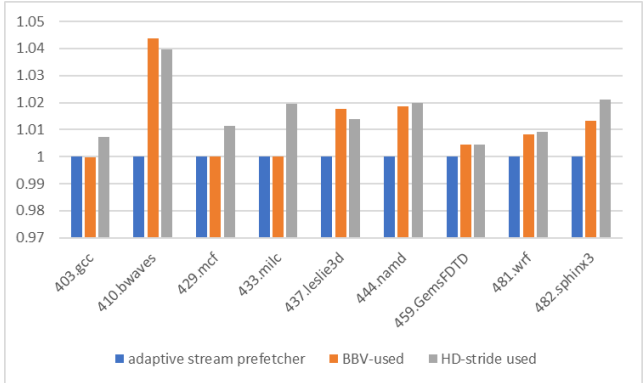

図 3　read hit rate on L3 cache(normalized)

キップし，その後の 1G 命令を実行する．ストリームプリフェッチャは L3 キャッシュにのみ適用し，他のキャッシュではプリフェッチを行わない．

過去のカウンタは全て保持するものとする．閾値は比較している 2 カウンタの総カウント数の合計の割合である．カウンタの比較と出力を行う間隔は 10M 命令とする．なお，2 次ストライドの検出においては命令のリタイア時に行うため，メモリアクセス命令はインオーダに処理される．

カウントは過去の全エントリを記録し正確に行う．2 次ストライドに関して，追跡するストリームの数の最大値は 64 エントリとする．2 次ストライドが検出できるようなストリーム同士の間に，無関係のストリームが 64 本以上現れることは非常に少ないため，この制限によって性能が低下することはほとんど無いことが分かっている．

6. 結果

フェーズ検出を用いないストリームプリフェッチャ（adaptive stream prefetcher），BBV によるフェーズ検出を用いたストリームプリフェッチャ（BBV-used），2 次ストライドによるフェーズ検出を用いたストリームプリフェッチャ（HD-stride used）のそれぞれを L3 キャッシュに適用した際の性能を比較する．ただし，有意な差が見られないベンチマークについては省略する．参考として，プリフェッチャを用いない場合（no prefetcher）の性能も表記する．

評価指標として，L3 キャッシュの読み出しにおけるヒット率および IPC（instructions per clocks）を用いる．数値の差が小さいため，フェーズ検出を用いないストリームプリフェッチャの数値で除算し正規化したものを併せて示す．ヒット率を図 2 および図 3，IPC を図 4 および図 5 にそれぞれ示す．

7. 考察

BBV によるフェーズ検出では性能が改善されていないものについても，2 次ストライドを用いたものでは改善されているものが存在することが読み取れる．メモリについて適応的な改善手法を取り入れる上では，アクセスされるアドレスという情報も有用であることが分かる．

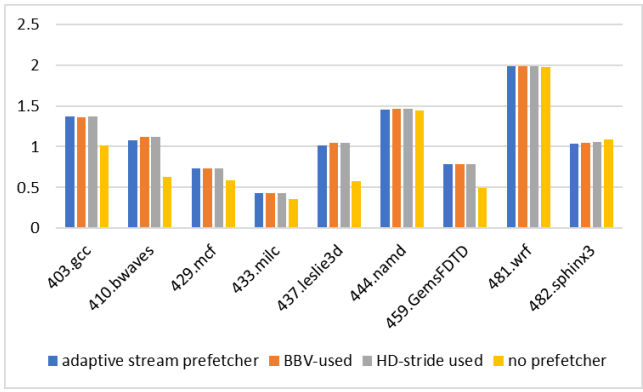

図 4　IPC

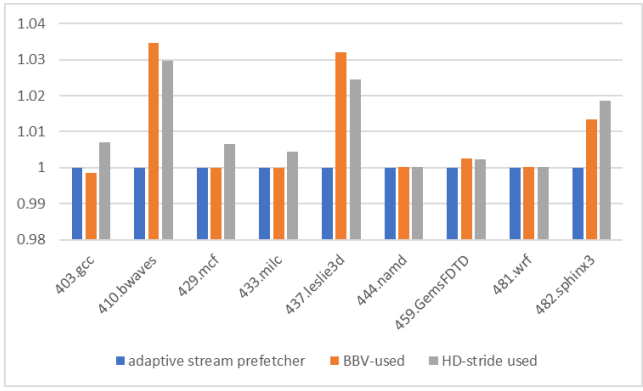

図 5　IPC(normalized)

482.sphinx3 では，プリフェッチをかけない場合が最も性能が高い．ただのストリームプリフェッチャをかけた場合，最も性能が低くなっていることから，不要なプリフェッチが悪影響を及ぼしていると考えられる．しかし，フェーズ検出を用いたプリフェッチャを利用したとき，ただのプリフェッチャよりも高い性能を示している．フェーズ検出に基づいてプリフェッチ量を増やしているにも関わらず性能が上がるということは，プリフェッチ量を増やすことが効果的であるフェーズが存在するということである．ここで BBV を用いたものと 2 次ストライドを用いた

ものを比較すると，2 次ストライドを用いたものの方が性能が良い．すなわち，2 次ストライドによるフェーズ検出の方が，無駄なプリフェッチを抑え，真にプリフェッチするべきフェーズを検出できていると言える．しかし依然としてプリフェッチしないものの方が性能が高いため，プリフェッチを増やすだけでなく抑制する手法についても検討し調べる価値がある．

更に性能を向上させる手法について検討する．フェーズとはプロセッサの動作の繰り返しからパターンを見出す概念である．フェーズについてメタ的に考えることも可能であり，フェーズの出現パターンから次にどのようなフェーズが現れるか予測する手法が考えられる．BBV によるフェーズ検出とプロファイラによる解析，ソースコードの埋め込みを組み合わせて次のフェーズを予測する手法は既に存在し [14]，2 次ストライドによる検出についても同様に予測機構を作ることが可能である．

容量的なオーバーヘッドについて考察する．BBV を厳密に正しくカウントするためには，ベーシックブロックを検出する機構を搭載した上で数百から数千のエントリを持つカウンタを保管するインターバル数だけ用意しなければならない．一方，2 次ストライドを用いた場合，ストリームプリフェッチャで用いられる機構と全く同じものを用意すればよく，カウンタのエントリ数も追跡する範囲で十分である．既にある機構を再利用できるために複雑になりすぎないことも利点である．更にオーバーヘッドを削減する方法として，（2 次の）ストリームと見做すために必要なシーケンシャルアクセスの個数を増やすことで雑多な検出を抑え，2 次ストライドの下位ビットを切り捨てるなどしてカウンタをまとめることが考えられる．

8. おわりに

本論文ではフェーズを用いたストリームプリフェッチャの改良について述べた．フェーズごとにプリフェッチするべき量をカウントによって調べ，プリフェッチが有効なフェーズでは多くプリフェッチを行うという手法により，最大で 3%程度の性能改善が見られた．今後の課題として，複数のパラメータを学習するなど，より複雑な学習を行う場合について調査することが挙げられる．また改良として，フェーズの現れ方のパターンから次のフェーズを予測する，コード解析等の他の解析と組み合わせる，などが考えられる．

参考文献

[1] Jain, A. and Lin, C.: Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement, *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 78–89 (online), DOI: 10.1109/ISCA.2016.17 (2016).

[2] Wu, C.-J., Jaleel, A., Hasenplaugh, W., Martonosi, M., Steely, Jr., S. C. and Emer, J.: SHiP: Signature-based Hit Predictor for High Performance Caching, *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, New York, NY, USA, ACM, pp. 430–441 (online), DOI: 10.1145/2155620.2155671 (2011).

[3] Jaleel, A., Theobald, K. B., Steely, Jr., S. C. and Emer, J.: High Performance Cache Replacement Using Re-reference Interval Prediction (RRIP), *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, New York, NY, USA, ACM, pp. 60–71 (online), DOI: 10.1145/1815961.1815971 (2010).

[4] Smith, A. J.: Cache Memories, *ACM Comput. Surv.*, Vol. 14, No. 3, pp. 473–530 (online), DOI: 10.1145/356887.356892 (1982).

[5] Hind, M., Rajan, V. and Sweeney, P. F.: Phase detection: A problem classification, Technical Report 22887, IBM Research (2003).

[6] Sherwood, T., Perelman, E. and Calder, B.: Basic block distribution analysis to find periodic behavior and simulation points in applications, *Proceedings 2001 International Conference on Parallel Architectures and Compilation Techniques*, pp. 3–14 (online), DOI: 10.1109/PACT.2001.953283 (2001).

[7] Sherwood, T., Perelman, E., Hamerly, G. and Calder, B.: Automatically Characterizing Large Scale Program Behavior, *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS X, New York, NY, USA, ACM, pp. 45–57 (online), DOI: 10.1145/605397.605403 (2002).

[8] Sherwood, T., Sair, S. and Calder, B.: Phase Tracking and Prediction, *Proceedings of the 30th Annual International Symposium on Computer Architecture*, ISCA '03, New York, NY, USA, ACM, pp. 336–349 (online), DOI: 10.1145/859618.859657 (2003).

[9] Tcheun, M. K., Yoon, H. and Maeng, S. R.: An adaptive sequential prefetching scheme in shared-memory multiprocessors, *Proceedings of the 1997 International Conference on Parallel Processing (Cat. No.97TB100162)*, pp. 306–313 (online), DOI: 10.1109/ICPP.1997.622660 (1997).

[10] Gill, B. S. and Bathen, L. A. D.: Optimal Multistream Sequential Prefetching in a Shared Cache, *Trans. Storage*, Vol. 3, No. 3 (online), DOI: 10.1145/1288783.1288789 (2007).

[11] 渋江陽人，野村隼人，入江英嗣，坂井修一：ストライドアクセスの階層構造に着目したフェーズ検出，電子情報通信学会技術研究報告= IEICE technical report : 信学技報， Vol. 116, No. 510, pp. 9–14（オンライン），入手先 ⟨http://id.ndl.go.jp/bib/028102070⟩ (2017).

[12] Tajik, H., Donyanavard, B. and Dutt, N.: On Detecting and Using Memory Phases in Multimedia Systems, *Proceedings of the 14th ACM/IEEE Symposium on Embedded Systems for Real-Time Multimedia*, ESTIMedia'16, New York, NY, USA, ACM, pp. 57–66 (online), DOI: 10.1145/2993452.2993566 (2016).

[13] 塩谷亮太，五島正裕，坂井修一：プロセッサ・シミュレータ「鬼斬弐」の設計と実装，先進的計算基盤システムシンポジウム SACSIS2009， Vol. 2009, No. 4, pp. 120–121 (2009).

[14] Zhang, W., Li, J., Li, Y. and Chen, H.: Multilevel Phase Analysis, *ACM Trans. Embed. Comput. Syst.*, Vol. 14, No. 2, pp. 31:1–31:29 (online), DOI: 10.1145/2629594 (2015).