社団法人　情報処理学会　研究報告
IPSJ SIG Technical Reports

2006－ＳＬＤＭ－126（３）
2006／10／26

CDFGからの複合演算抽出手法

白井 雄一† 西沢 政則† 大角 嘉蔵† 西門 秀人† 加藤 俊之††
山内 寛紀† 小林 士朗†††

†立命館大学理工学部 〒525-0123 滋賀県草津市野路東 1-1-1
††日本ケイデンス・デザイン・システムズ社 〒222-0033 神奈川県横浜市港北区新横浜 3-17-6
†††旭化成株式会社研究開発本部 〒243-0021 神奈川県厚木市岡田 3050
E-mail: †{re005017, ro007002, ro000029}@se.ritsumei.ac.jp

あらまし　本稿ではSUFFIX TRIEを用いたCDFGからの複合演算抽出手法を提案する．DSPの積和演算器に代表されるように，特定用途向けのLSIには高性能化を狙った専用回路が搭載される事が一般的，専用回路化すべき演算パターンを効率的に求める事ができる．我々はCDFGを用いたC言語からのLSI自動生成システムの研究を進めており，本手法を用いる事で自動的に高性能・高効率なLSIの生成が可能となる．

キーワード　CDFG, SUFFIX TRIE , DSP, High-Level syntheses

Combine operation pattern extraction from CDFG for DSP generation

Yuichi Shirai† Masanori Nishizawa† Yoshizo Osumi† Hieto Nishikado†
Toshiyuki Kato†† Hironori Yamauchi† Shiro Kobayashi†††

† College of science and engineering, Ritsumeikan University 1-1-1 Nojihigassi, Kusatsu-shi, Shiga, 525-0123 Japan
†† Cadence Design Systems, Japan 3-17-6 Sinyokohama, Kouhoku-ku, Yokohama-shi,Kanagawa 222-0033 Japan
††† Central R&D Administration, Asahikasei 3050 Okada, Atsugi-shi,Kanagawa 243-0021 Japan
E-mail: †{re005017, ro007002, ro000029}@se.ritsumei.ac.jp

Abstract In this paper, we propose a method to extract frequent operation patterns from the CDFG used to SUFFIX TRIE. Represented for accumulator in DSP, almost specific LSI has exclusive calculator for high performance. To use the method, you can find combined operation pattern that you have to make exclusive calculator easily.Our group is working on development of DSP automatic generation systems. Therefore, we developed a system to generate CDFG based on C language scripts, and to extract frequent operation patterns from CDFG.

Keyword CDFG, SUFFIX TRIE, DSP, High-Level syntheses

1. はじめに

現在，携帯電話をはじめとする組み込みシステムの発展が著しい．各製品の多機能化が進み開発サイクルも短くなっている．そのためLSIの開発負担の増加が問題となり,抽象度の高いLSI開発手法の研究が盛んとなってきている．現在では，C言語の動作記述から回路を自動生成する動作合成が開発現場で用いられる様になってきている．しかし，既存の手法[1]ではHW処理部とSW処理部とを人が指定しなければならないためSoCやDSP(Digital Signal Processor)等のLSIの開発に用いる事は出来ない．現在，本研究室では特定用途向けであるプロセッサ,DSPを自動合成のターゲットとしてHW自動合成システムの研究を進めている．

本研究室の提案するDSP自動合成システムをはじめ，HW自動生成手法にはCDFGを用いる手法が多く存在する[2][3]．しかし，特殊演算器等のHW処理は人が指定しなければならない．そこで今回CDFGから頻出パターンを発見し頻出演算パターンを求める手法を提案する．この手法を用いる事で自動的に複合演算を求める事ができHW/SW分割手法に用いる事ができる．2.で本研究室の提案するDSP自動合成システムの概要を述べる．3.でCDFGについての説明を行い，本手法で用いるSUFFIX TRIEについて4.で紹介する．そして５.にて本手法の説明を行い，実装結果を6.で紹介する．

2. DSP 自動合成システム

提案する DSP 自動合成システムは DSP ハードウェアの設計期間の短縮を目的としたものである．設計者は所望する機能が記述されたＣ言語記述を当システムに与える．そうすれば，本システムが記述されたアルゴリズムに最適な DSP のハードウェアの構成と実行コードを出力する．

図 1 に DSP 自動合成システムの処理フローを示す．当システムは，大きく分けて CDFG 生成部と CDFG 最適化部、ハードウェア生成部の３つからなる．CDFG 生成部では，入力されたＣ言語記述を解析し、構文解析木を作る。次に，この構文解析木の依存解析を行う．その結果から CDFG を生成する。CDFG 最適化部では、CDFG から性能向上が多く見込める演算パターンを発見し複合演算を抽出する。この複合演算の情報はプロセッサ内の演算器を生成するときに必要となる。そして、ハードウェア生成部では CDFG と複合演算の情報を元にハードウェア構成を決定する．出力は HDL とする。また、そのプロセッサ上で実行するための実行コードも同時に生成する。

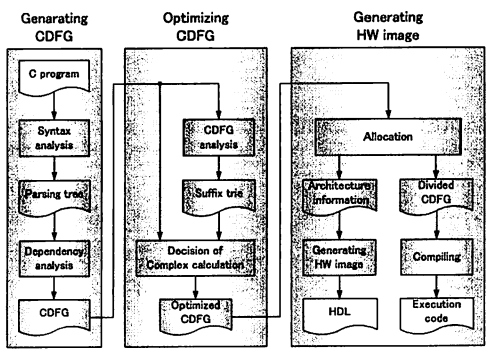

図 1 DSP 自動合成システムの処理フロー

3. CDFG

3.1. CDFG

DFG(Data Flow Graph)とはデータの流れを表したグラフの事をいう．DFG の例を図 2 に示す．円で囲まれた演算子は演算処理を表し，各円を繋ぐ線はデータの流れを表している．この DFG は A=(B+C)+D を表している．CDFG とは，この DFG に,分岐や繰り返し等の制御情報を加えたグラフの事をいう．一般的な CDFG を図 3 に示す．ここでの三角形は分岐処理を表しており，制御変数'x'の値によって異なる演算処理が行われる．この CDFG は 'x'が 0 なら a=(b-d)+c; 'x'が 1 なら a=b+d; の処理が行われる事を示している．CDFG には特定の書式は決まっておらず多くの種類の CDFG が提案されている．本研究室でも DSP 自動合成に適した独自の CDFG を提案しており，次項に本研究室の提案する CDFG の説明を行う．今回の複合演算抽出手法では，この CDFG を用いているが，本稿の提案手法は CDFG 全てに用いることが可能である．

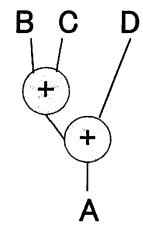

図 2 DFG

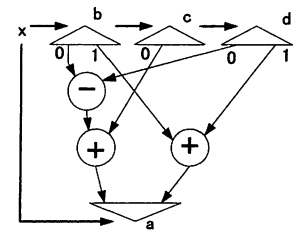

図 3 CDFG

3.2. 提案する CDFG

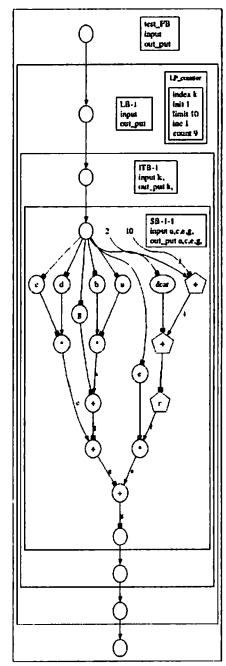

図 4 提案 CDFG

本研究室で提案する CDFG の例を図 4 に示す．図 4 における四角の枠は階層を表している．Ｃ言語では If 文や for 文などの制御処理は入れ子状に幾つでも重ねる事が可能である．（いくらでもネストが可能）そこで，この CDFG では制御を階層構造を用いて表している．これにより CDFG とＣ言語記述との対応関係が理解しやすいだけでなくマクロからミクロの可変の大きさで並列性を抽出することができる．各階層の右上に四角い枠に囲まれた表記がある．各階層には処理の種類によってラベルを付けている．関数全体は FB (function block).分岐処理は SLB(selection block).ループ処理部は LB（loop block）及びループ毎の処理を ITB (iteration block).演算部を SB (statement block)としている.LB の右上にはラベル名と同時にループ条件を表記する.演算処理を表す SB では，演算子を表すノードに円と五角形が存在している．これはアドレス演算部とデータ演算部の CDFG を区別するためである．五角形のノードで表現された部分はアドレス演算部となっている．丸いノードで表現された演算はデータ演算である．アドレス演算部とデータ演算部を分離しているため処理の流れを理解しやすい．また DSP の様にアドレス計算器を持つアーキテクチャに対しても最適な情報を得ることが可能となっている．

本研究室ではこのCDFGの自動描画システムの開発を行っている．そのためC言語を入力として図4の様なCDFGを自動的に描画することができる．

4. SUFFIX TRIE

SUFFIX TRIEとは語句解析や辞書式圧縮法に用いられている木構造である．語句や文章を入力としており，語句に存在する全てのパターンを羅列する．それに加えてパターンの出現回数を表す．具体的なSUFFIX TRIEの生成手法は[4][5]を参照のこと．

図5に文字列[abbabc]から生成したSUFFIX TRIEを示す．図5において、円の中に入っている数字は出現回数を表している。円の中に数字が入っていないものは1度だけ出現したパターンである。入力された文字列”abbabc”には”a”が二回、”b”が三回、”ab”が2回存在している事が分かる．

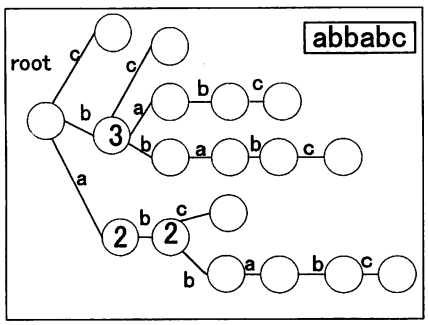

図 5 SUFFIX TRIE の例”abbabc”

5. 複合演算抽出手法

本章ではCDFGからのSUFFIX TRIEを用いた複合演算手法を提案する．ここで用いるCDFGは3.で説明したCDFGを用いるが，CDFGと呼ばれるものであれば本手法を用いる事は可能である．本手法では，まず入力となるCDFGからSUFFIX TRIEを作る．そこで，CDFG上の演算の出現パターン及び出現回数（頻出度）が解る．そして,その SUFFIX TRIE を基に複合演算を決定するのである．複合演算を決定した後,CDFGの頻出演算パターンを複合演算に置き換えることにより，複合演算器を持つHWの合成が可能となる．

5.1. 頻出パターン抽出手法

頻出演算用に専用演算器を持つためには，頻出演算パターンを知る必要がある．本提案手法では，SUFFIX TRIEを用いて，頻出パターンを抽出する．前述の通りSUFFIX TRIEは文字列に用いられている木構造であるため,本手法ではSUFFIX TRIEをCDFG向けに拡張して用いている．入力を文字列ではなく，グラフ構造に対応したのである．CDFGからSUFFIX TRIEを描く事でCDFG上に存在するパターン全てが羅列されるのだ．

5.1.1. 演算部(Statement Block)

CDFGは演算部と制御部に分かれるが,CDFG上の制御部には演算は無い．そのため，まずは演算部（SB）においてSUFFIX TRIEを生成する．

図6に,図4の演算部(SB_1-1)から生成したSUFFIX TRIEを示す．左上にSB 1-1とラベルが付加されている．Rootと書かれているノードは開始点を表す．エッジの添え字は存在する演算子を表す．2つ以上のノードの並びは，出現パターンを示している．ノードの中に入っている数字は出現回数を表している．ノードの下に表示されているIDは手動で指定する際などに使用する．図6には対象となるCDFGに存在する演算パターン全てが列挙されている．また”++”及び”+*”のパターンが3回存在していることも解る．

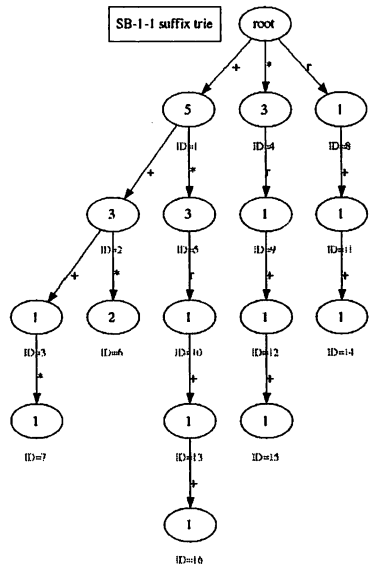

図 6 SB レベルの SUFFIX TRIE

5.1.2. 制御部

今回用いたCDFGでは制御処理を階層として表しているため，最下層は全て演算部（SB）である．そのため，処理の手順としては演算部のSUFFIX TRIEをつくり，制御処理を加える事で全体のSUFFIX TRIEを得る．制御部にはループや分岐がある．ループ処理ではループ回数をCDFGが保持しているため，ループ回数を出現回数に乗算する．分岐処理では各分岐のSUFFIX TRIEを加算したあと,分岐数に応じて出現回数を除算する．また，同階層に複数のSUFFIX TRIEがあれば加算する．この加算は数値の加算ではなく木構造の加算なので木の形が変化する事もある．図7に図4のCDFGから生成した全体のSUFFIX TRIEを示す．ループ回数が乗算されたため出現回数が大きくなった事がわかる．

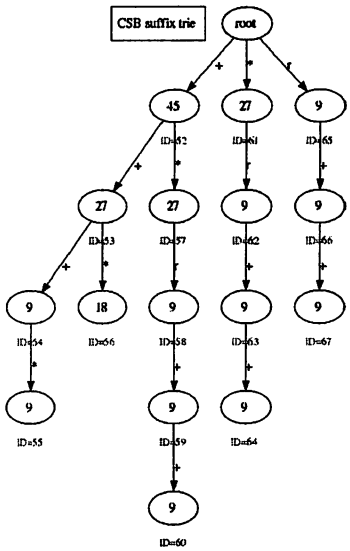

図 7 全体の SUFFIX TRIE

5.2. 複合演算決定

SUFFIX TRIE を用いて頻出パターンを知る事ができた．この結果から複合演算を決定する．SUFFIX TRIE には演算パターンと演算パターンの存在回数が出ているので，ここから「百回以上存在する 3 演算のパターン」や「3 演算の最も多いパターン」などの条件を与える事で簡単に求める事ができる．

6. 結果

今回，C 言語記述を入力として，CDFG 及び SUFFIX TRIE を生成するシステムの試作を行った．本節ではその結果を紹介する．今回，評価として入力した C 言語記述は Dct（離散コサイン変換）である．Dct は最も有名な信号処理の一つであるために適していると考えた．入力した dct の記述を以下の図 8 に示す．C 言語記述の上部にある配列には，コサインの値が初めから入っている．そうする事でコサイン処理をデータ呼び出しで実現している．

この記述から描いた CDFG の一部を図 9 に示す．この部分は C 言語記述の最も内側のループでの処理を示している．この CDFG から SUFFIX TRIE を描き SB レベルでの SUFFIX TRIE を得る．その SUFFIX TRIE を図 10 に示す．ここから，制御部の処理を行い，DCT 全体の SUFFIX TRIE を生成する．この結果を図 11 に示す．

```
void main(void)
{

double cos_table[32]={1.0, 0.9807853, 0.92387953, 0.831469612, 0.70710678,
                      0.555570233, 0.3826834, 0.1950903, 0.0,
                      -0.1950903, -0.3826834, -0.555570233, -0.70710678,
                      -0.831469612, -0.92387953, -0.9807853, -1.0,
                      -0.9807853, -0.92387953, -0.831469612, -0.70710678,
                      -0.555570233, -0.3826834, -0.1950903, 0.0,
                      0.1950903, 0.3826834, 0.555570233, 0.70710678,
                      0.831469612, 0.92387953, 0.9807853};*/

double cv[7]={0.70710678,1.0,1.0,1.0,1.0,1.0,1.0};
double cu[7]={0.70710678,1.0,1.0,1.0,1.0,1.0,1.0};
double dct_after[10][10];
double dct_before[10][10];

 int x,y,u,v;
 double sum;
 for(v=0; v<8; v=v+1){
          for(u=0; u<8; u=u+1){
    sum=0;
    for(y=0; y<8; y=y+1){
                    for(x=0; x<8; x=x+1){
                                sum =sum + dct_before[y][x] * cos_table[((2*x+1)*u)%32] *
cos_table[((2*y+1)*v)%32];
                    }
    }
          dct_after[v][u] = sum*cu[u]*cv[v]/4;

  }
 }
}
```

図 8 DCT 記述

6.1. CDFG

図 9 に CDFG の生成結果を示す．今回は一部しか掲載していないが，全体像での生成を行う事ができた．階層も表現されている．データ演算とアドレス演算との表現を変えてはいるが，理解が難しい事が解った．理解しやすいグラフに改良が必要と考えられる．しかしながら，このグラフは graphviz[6]というツールを用いて描いているため,この graphviz の性能に依存しており,多少入り組んだグラフになることは仕様となる．

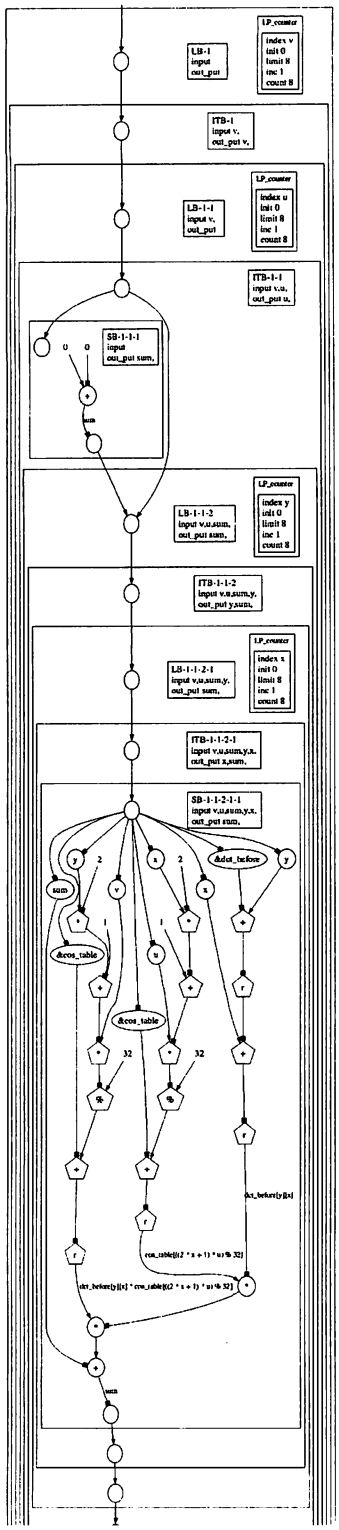

図 9 DCT の CDFG（一部）

6.2. SUFFIX TRIE

図 10 に演算部の SUFFIX TRIE の結果を示す．世代数は設定することが可能（今回は 5 世代）.満足な結果を得る事が出来た．この結果から，"+→r→*"や"*→+"といったパターンが多く存在している事が解るが演算部だけの結果では，出現回数は少なく違いが少ない．ここからループ回数乗算し，各部の SUFFIX TRIE を足し合わせていき，全体の SUFFIX TRIE を得る．この結果を図 11 に示す．この図では"*→＋→*"のパターンが 8192 回存在するなど，存在回数が非常に多くなり，複合演算の決定を行うことが出来る．ここまで出現回数が大きくなるのは，今回対象とした DCT が非常にループ処理の多いプログラムで合ったため出現回数が何度も乗算されたためである．この結果から出現回数はループ回数と非常に密接な関係があることがわかる．そのため，複合演算を決定する際に“何演算で何回以上出現”といった条件を出す際には，ループ回数と関連を持たせないと所望の結果が得られない可能性が高いと言えるのでは無いだろうか．

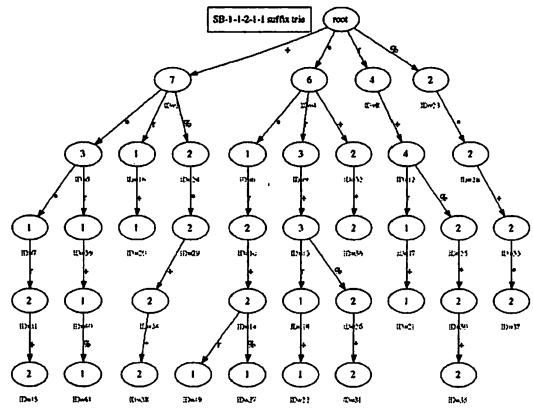

図 10 SB の SUFFIX TRIE(DCT)

7. まとめ

本稿では，CDFG からの SUFFIX TRIE を用いた複合演算抽出手法の提案を行った．C 言語などの高位言語からの LSI 自動設計システムでは，一度高位言語を CDFG に変換して，そこから LSI を設計していく手法が多く提案されている．そのため CDFG の中から頻出演算パターンを自動的に求める手法があれば，専用演算器の指定や，HW/SW コデザインにおける HW/SW 処理の分離に用いる事が可能であると考えている．本手法では従来，言語解析等の文字列にしか使われていなかった SUFFIX TRIE を CDFG という半構造データの領域に持ち込み，非常に効率的に頻出パターンの抽出を行っている．

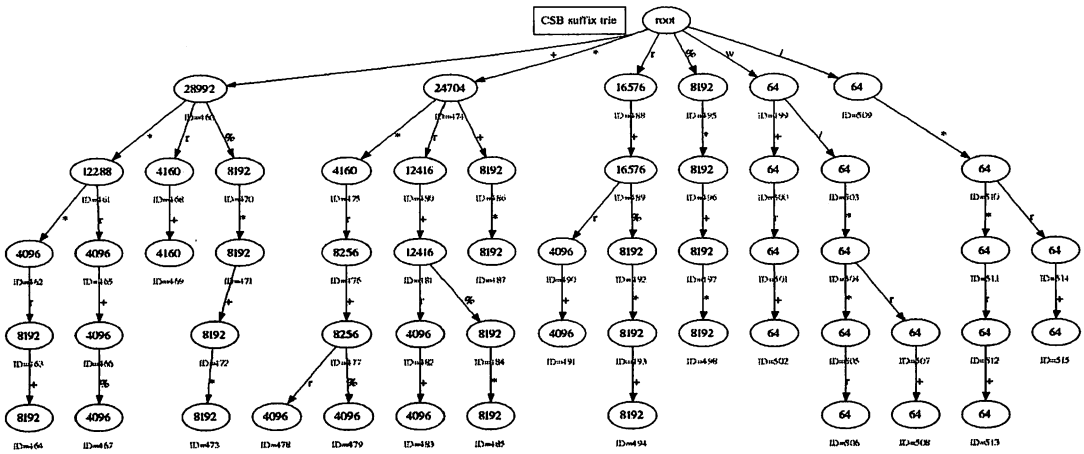

図 11 DCT の SUFFIX TRIE(全体)

今回はC言語記述を入力としてCDFGを生成するシステムと,そのCDFGからSUFFIX TRIEを生成するシステムの試作を行った.その結果C言語記述に出現する演算パターン全てを抽出する事が可能となり,出現パターンの出現頻度も求める事ができた.この結果から,容易に複合演算を求める事が可能である.

今後は,SUFFIX TRIEからの複合演算抽出を行っていく.出現回数は知る事ができたが各演算の重み付け等を行っていないので,正確にどの演算パターンをHW化すれば最も効率が良いか知る事ができない.また,どれくらいの性能向上が見込めるかの指標も求めて生きたいと考えている.

文　献

[1] D. Gajski, A. Wu, N. Dutt, and S. Lin, "High-level Synthesis: Introduction to Chip and System Design.", Kluwer Academic Publishers, 1992.

[2] 川田容子,戸川望,佐藤政生,大附辰夫,"動作記述からのデータフローグラフ生成手法,"電子情報通信学会技術研究報告, VLD, vol95(307), pp55-62, Nove.1995.

[3] 西口健一,石浦菜岐佐,西村啓成, 神原弘之, 冨山宏之, 高務祐哲, 小谷学, "ソフトウェア互換ハードウェアを合成する高位合成システムCCAPにおける変数と関数の扱い,"電子情報通信学会技術研究報告, ICD, Vol.105(446), pp.19-24(2005)

[4] 中西恒夫, 中野猛, 福田晃, "辞書式コード圧縮支援機構の遺伝的アルゴリズムによる最適化," 電子情報通信学会技術研究報告, ARC, Vol.2001(39), pp.19-24,(2005).

[5] Christopher W.Fraser, Eugene W.Myers, Alan L.Wendi, "Analyzing and Compressing Assembly Code", Proceedings of the ACM SIGPLAN '84 Symposium on Compiler Construction SIGPLAN Notices Vol.19, No.6, pp.117-121(1984)

[6] http://graphviz.org